MarketNet: Market-Based Protection of Information Systems

Y. Yemini, A. Dailianas, D. Florissi

Department of Computer Science
Columbia University,
450 Computer Science Bldg.
New York, NY 10027
Email: {yemini, apostolo, df}@cs.columbia.edu

G. Huberman

School of Business
Columbia University,
411 Uris Hall
New York, NY 10027
Email: gh16@columbia.edu

Abstract

This paper describes novel market-based technologies for systematic, quantifiable and predictable protection of information systems against attacks. These technologies, incorporated in the MarketNet system, use currency to control access to information systems resources and to account for their use. Clients wishing to access a resource must pay in currency acceptable to the domain that owns the resource. An attacker must thus pay to access the resources used in an attack. Therefore, the opportunities to attack and the damage that can be caused are strictly limited by the budget available to the attacker. A domain can control its exposure to attacks by setting the prices of critical resources and by limiting the currency that it makes available to potential attackers. Currency carries unique identifiers, enabling a domain to pinpoint the sources of attacks. Currency also provides a resource-independent instrumentation to monitor and correlate access patterns and to detect intrusion attacks through automated, uniform statistical analysis of anomalous currency flows. These mechanisms are resource-independent, and admit unlimited scalability for very large systems consisting of federated domains operated by mutually distrustful administrations. They uniquely establish quantifiable and adjustable limits on the power of attackers; enable verifiable accountability for malicious attacks; and admit systematic, uniform monitoring and detection of attacks.

1. Introduction

Protecting large-scale information systems remains an elusive challenge of ever-growing importance and complexity. Exposure to insecurities and the opportunities available to attackers are increasing with the growth in the range of resources, scale, complexity, and operations management practices of different domains. Current information systems enable attackers to pursue virtually unlimited attempts to compromise a system; they involve ad-hoc instrumentation to monitor resource access and manual correlation of these access logs to detect intrusion; and they leave attackers completely unaccountable to abuses and crimes that they commit.

Rapid changes in technologies increase the vulnerability to attackers. First, at present protection technologies are specialized to each component. A minor insecurity in a new component can propagate substantial exposure to other components. Thus, insecurities can be formed non-monotonically; i.e., a system that is secure can be rendered insecure by the addition of a single component. Second, the combinatorics of interactions between new components and existing ones increases exponentially, creating ample possible insecurities. Third, in the absence of a unifying security architecture it is practically impossible for component vendors, or domain administrations to accomplish a coordinated protection.

Domain administrations are thus increasingly exposed to security risks and are unable to control, bound or even assess this exposure. They require expert manual labor to monitor and correlate access anomalies and detect an attack, typically through off-line non-real-time processes completed hours or days after the

attack has been completed. And even when an attack is detected, identifying the source accountable for it can be virtually impossible and requires complex ad-hoc collaborations of multiple expert police forces. And this potential exposure and complexity of protection increases with each change in resources or their configurations.

MarketNet uses market-based mechanisms to provide a novel approach to the protection of large-scale information systems. In MarketNet, resources are instrumented to use currency for access control and monitoring. Clients must pay resource managers, using appropriate currency and prices, to gain access to respective resources. An attacker is limited by its budget in gaining access to resources and in creating damage. A domain administration can control access to its resources and establish quantifiable limits on its exposure to attacks by adjusting prices of critical resources and controlling the availability of currency to potential sources of attacks.

Currency flows provide uniform resource-independent instrumentation to monitor and correlate access patterns and to detect anomalies. This enables the development of uniform resource-independent intrusion-detection mechanisms entirely based on the statistics of currency flows. Intrusion-detection can be thus automated and accomplished in real-time with an attack. Furthermore, currency carries unique identifiers. A domain maintains full accountability of the entities to which currency has been allocated. A domain can account for sources of each access to its resources. In particular, once an attack has been identified a domain can establish verifiable proof of accountability in tracing its sources.

MarketNet mechanisms are structured to admit unlimited scalability and enable protection among mutually distrustful domains organized in a large scale federated system. These protection mechanisms, furthermore, are entirely independent of the underlying resources and can thus be retrofitted into an existing system with minor adaptation of its components.

This paper provides an overview of the MarketNet architecture, mechanisms and operations.

2. MarketNet architecture and mechanisms

2.0 MarketNet Architecture

MarketNet introduces a distributed protection middleware infrastructure, the Resource Access Layer (**RAL**), overlaid on existing infrastructure (Figure 1). RAL includes several mechanisms. Resource managers are responsible to set the price for a resource, collect payments for its access and deposit revenues with the bank server of the respective domain. Client managers are responsible to manage client budget, obtain pricing information and pass respective payments required to access services used by the client. Bank servers provide accounting, clearing and monitoring of currency flows. Price directories provide pricing information. These mechanisms are depicted in Figure 1 below.

In a typical scenario, depicted in Figure 1, a client belonging to domain X wishes to access a resource belonging to domain Y. The client needs to first obtain currency acceptable to domain Y. The client manager obtains respective pricing information and issues a request to the bank server of domain X to provide it with currency for domain Y. The bank server of domain X must obtain currency issued by domain Y and credit the account of domain Y with a respective central bank for this amount. The two domain bank servers pursue secure transactions with the central bank to accomplish this. Once the client manager obtains respective currency from its bank server, it can proceed to execute accesses to the service. Each access will incur a payment collected by the server manager.

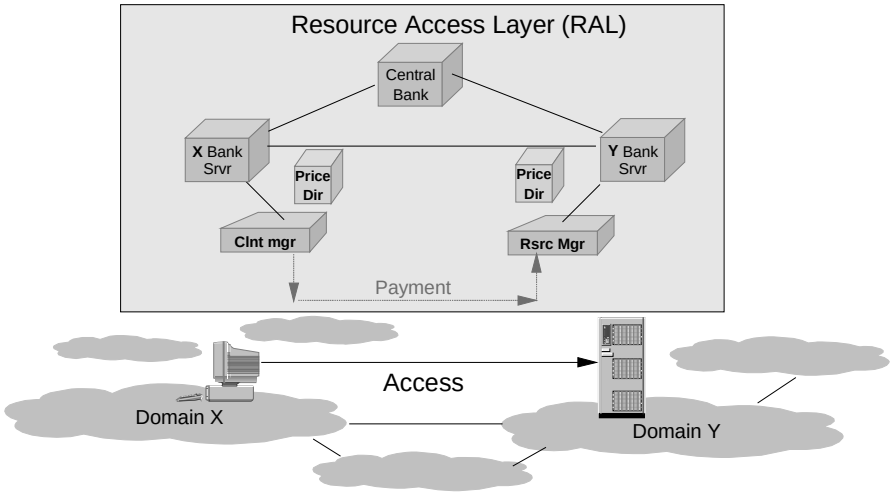


Figure 1: Overall Architecture of MarketNet

It is important to note several salient features of this architecture.

- The RAL mechanisms provide incremental extensions of existing components and systems.
- The client and resource managers provide uniform (i.e., resource independent) and minimal extensions of existing software components to control access.
- All transactions involving currency flows between managers and their bank servers and between bank servers are secured through encryption and authentication.
- The overheads involved in converting currency among domains and in allocating currency to a client can be minimized through caching of currency. For example, the bank server of domain X can cache sufficient currency of domain Y in anticipation of requests by clients in its domain.
- Once a client obtains currency, the payment to resource managers involves very minimal overhead.

Thus, the RAL provides in effect a distributed secure access management kernel that is independent of underlying resources.

2.1 Currency domains organize global protection

Resources and clients in MarketNet are organized in currency domains. Resources include physical resources such as CPU cycles, storage, bandwidth, I/O devices, or sensors as well as higher-level software services such as file storage, name service, database, or web service. A currency domain establishes access protection for a group of resources. It provides administrative infrastructure for imposing domain-level protection policies covering pricing of critical resources, assignment of budgets to internal clients, limitation of access by external domains, monitoring access to detect intrusion attacks and activating responses to attacks.

Domains use special currency to provide unified, scalable access to their services. The currency is uniquely identified by a *currency ID*. This establishes full accountability in the use of resources by tracing access to resources back to the holder of the currency. To gain access to the resources in a domain, clients first have to exchange currency of the target domain for their own. Currency of a domain gives the holder the right to access any of the resources in the domain, providing unified, scalable access. Finer access control at the resource level is achieved through the pricing and usage-monitoring mechanisms presented in Sections 2.2 and 2.4. The currency of a domain encapsulates domain-level protection policies set by the domain. Specifically, domains control who can acquire their currency, along with the total currency outflow, the rate of currency outflow and other parameters, imposing strict domain-controlled limits on the access and attack power of any entity wishing to access the domain resources.

2.2 Prices/budgets establish dynamically adjustable access control

Prices of resources along with available budgets of clients establish a dynamically tunable access control mechanism; provide the means for optimized load redistribution and graceful degradation upon loss; and impose quantifiable dynamically adjustable limits on the exposure to attackers.

Each resource in a domain is priced in terms of the domain currency. This price is advertised in respective price directories of the RAL. Prices are dynamically updated to reflect various operation parameters such as access control policies and changing demand for a resource. The combination of prices and budgets available to clients provides a fine granularity, dynamically adjustable access control mechanism. Limiting access to a specific set of clients can be achieved by raising the prices to higher levels, guaranteeing that only qualified clients (those that have sufficient budget) can access them. Furthermore, currency identifiers enable additional price discrimination techniques. Budget and price discrimination can achieve a continuous spectrum of limits imposed on the use of a resource, based on the source domain of a request.

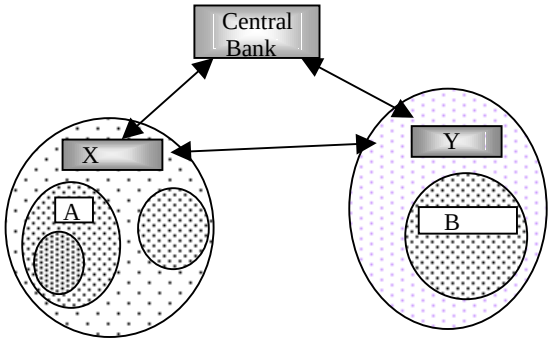
The pricing mechanism can also be used to reflect resource unavailability due to congestion or loss. A loss of a resource reduces the supply, thus automatically causing clients to redirect their demands to backup replicas. Reduced availability results in rising prices of the replicated resources. Rising prices create a natural selection process where applications automatically adapt to resource availability and obtain access to alternate resources according to their intrinsic priority captured by their budget. High-priority clients can apply their budget to continue and obtain high QoS, while low priority clients are priced-out. Thus a loss results in graceful selective degradation of services that optimizes the balance between available resources and demands.

Furthermore, prices can force the operation of resources within a "*desirable*" region of operation. The *desirable* region of operation is resource-dependent, and in general refers to the region of operation specified by the resource manager, where specific QoS constraints or other considerations are satisfied (e.g., the average incoming rate to a switch should be controlled to provide low delays and loss). Assume the purpose of attacking a resource is to move it to an "undesirable" region of operation. Then the price of the resource should reflect its reluctance to operate in that region. Should the attacker or coalition of attackers desire to sustain the attack, they would see a continuously increasing price to access the resource, forcing them to exhaust their budget at an increasing rate to sustain the attack. The pricing mechanism in this case provides a means to convert a "fixed" budget (belonging to a specific client or a coalition of clients potentially residing in different domains), to a much lower "effective" budget. Knowledge of the specific pricing policy can provide analytical upper bounds on the duration of attacks achievable by given collective budgets.

2.3 Setting Quantified Limits on The Power of Attackers

MarketNet limits the power of attackers by their available budget. An attacker can gain access to resources only to the extent that his budget permits it. Furthermore, with each access required for an attack, the remaining budget and with it the power of the attacker decreases.

Thus, *enforcement* of budgets (i.e., guarantees that no client or application can spend more than their budget) is a very powerful tool for limiting attacks and damages. MarketNet pursues hierarchical budget enforcement by organizing network entities (i.e., resources, clients, or subdomains) in hierarchically nested domains; each domain has a bank that controls the budget usage of the entities inside the domain. For example, in the figure to the right, the budget of a user in domain **A** is controlled by the bank of this domain, the budget of the entire do-



main **A** is controlled by the domain **X** bank, and the budget of the domain **X** is controlled by the central bank. Even if an intruder conquers the **A** bank, his/her budget (and power of attack) cannot exceed the budget for the whole **A** domain. Similarly, even if the **X** bank is conquered the amount of damage the intruder can do is limited by the **X** budget, enforced by the central bank. Moreover, an intruder that conquered the **X** domain is not only limited in its power to attack other domains outside the scope of **X**, but also in its ability to attack interior domains. For example, to access any resources of the nested domain **A**, the intruder must still obtain sufficient currency of **A**. The amount of currency of **A** maintained in the bank of **X**, or made available to it, is strictly controlled by **A**.

The total exposure of a domain to external attacks is thus limited by the total budget that it maintains in external domains as well as the rate at which it accepts payments for services from external clients. A domain can tune both parameters to control its exposure.

Exchanges of currency between domains are performed by the banks at the outermost enclosing domains. For example, a client from domain **A** that wishes to access services of **B** needs to pursue an exchange conducted between the **X** bank and **Y** bank. This restriction is imposed for scalability and authentication reasons. Organization of resources in currency domains provides the means to scaleably limit the spread of faults or attacks and localizes their effects. For example, assume an intruder has conquered the whole **X** domain. Once any domain detects this, the information can be rapidly propagated to other domains and the currency of **X** can be declared invalid until appropriate action is taken to restore normal operation of the **X** domain.

2.4 Monitoring and detection of intrusion attacks

The use of currency to access resources provides resource managers with a uniform instrumentation to measure resource access and to identify anomalous patterns of access. Currency IDs, the unique identifiers carried by currency, establish liability in the use of resources by enabling one to determine the entity responsible for given resource accesses. The combination of resource access monitoring, and correlation of access and entities through currency IDs is used to identify and isolate attack sources.

Monitoring of currency flows is used to detect attacks. Monitoring can be done in much the same fashion as in typical transaction processing systems. Currency flows provide a good way to model temporal behaviors of clients and patterns of resource access to classify activities into those that are legitimate and those that seem suspicious and hence warrant further inspection and authorization. Once an attack has been identified, currency IDs are used to identify and isolate the source of the attack, without affecting the operation of users in legal domains. In case a whole currency has been conquered and the breach is detected, the currency of that domain can be declared invalid, limiting the spread of faults or attacks, until appropriate action is taken to restore normal operation.

3. Attack Sources

MarketNet is designed to detect, react to, and prevent attacks to systems. The goal of this section is to clearly define what are the sources of attack to computer systems and what MarketNet can do in each case.

3.1 Vulnerability of the MarketNet Infrastructure

A system is as safe as the protection subsystem it uses. The natural question is therefore how safe the MarketNet infrastructure is itself and how does it prevent undesirable protection leaks. There are two po-

tential attacks of interest: (1) circumvention of MarketNet, (2) tempering with the MarketNet infrastructure.

Figure 1 illustrates how the Resource Access Layer (RAL) middleware introduced by MarketNet is overlaid on existing infrastructure. What if an attacker decides to manipulate resources by circumventing the MarketNet APIs? For example, a process may decide to directly allocate memory at a particular node using the OS API. MarketNet should encompass such allocations, which may otherwise undermine or lower its own allocations. Therefore a crucial task in the MarketNet infrastructure is to enforce that resource manipulations may only happen via well-defined MarketNet APIs. The system that installs MarketNet must disable any "back-door" APIs that may enable attackers to directly manipulate resources.

Another option is to use the MarketNet infrastructure itself to attack. The best way to accomplish this is to tamper with the protocols and processes to gain illegitimate access to resources. The most obvious case of such attacks is to abuse bugs in the implementations. This is a problem with any software system and we are using sophisticated Software Engineering technology that minimizes such bugs.

More peculiar to MarketNet is to counterfeit, duplicate, or steal money. For example, some agent may discover a way of generating money that looks legal either from scratch or by duplicating money already in the system. Alternatively, it may attack the communication channels to acquire digital money in transit. Illegitimate wealth can become a security threat for any system relying on MarketNet for security. The protocols in MarketNet outlined in Section 4.1 are designed to virtually eliminate the possibility of such occurrences.

3.2 The Power of Wealth in MarketNet

Money concentration, even if legitimate, is undesirable due to the potential attack power it represents. Rich agents constitute a potential attack threat. They are also prone to attacks by intruders who are attracted by the attack power they would gain by taking over the rich agents.

The owner of a popular resource or service will accumulate unbound wealth over time if the resource generates revenues in excess of its operational costs. One option would be to impose restrictions on the wealth a single agent can accumulate. This would limit the power of any single entity to attack, but it would also reduce the incentive of resource managers to provide services. An alternative is to impose restrictions on the rate at which the accumulated wealth can be used to purchase other resources or services. This is made possible by the intervention of the banking infrastructure in the transactions between clients and resource managers. These issues are further examined in Section 4.2.

3.3 Abusive Consumption of Resources

Another potential attack is to abuse consumption of resources. It may take place in multiple ways. Firstly, one may simply conquer the resource and use it without paying dues or even profiting by re-selling services. Such situation must be prevented at all costs by enforcing that only MarketNet APIs can be used to access and manipulate resources.

Secondly, an agent may acquire cheaper or even nominal access to resources and either use them or resell them for profit. The MarketNet APIs prevent such situations by requesting immediate payment for resource consumption.

3.4 Non-for-profit Attacks

Some attackers are willing to disrupt or deny services to other users even when they do not profit directly from the attack. Open systems are susceptible to such attacks, and have proven particularly vulnerable to them. Part of the reason is that there is no uniform and scalable way to protect against them. The famous

denial-of-service attack is a good example of this category of attacks. The goal of such an attack is to bring down some service or resource by overloading it with requests for service, (e.g., requests for access to web sites, consumption of memory cache or processor cycles). Other users perceive the resource or service as unavailable or unable to deliver the expected performance.

Non-for-profit attacks usually manifest themselves in two ways. The first form of attack is to overload resources in the system, like in the denial-of-service case. MarketNet uses its price adjustment mechanism to cope with such situations. Increases in demand force prices to go up and eventually the attacker will run out of budget. The effectiveness of this mechanism may be emphasized if discriminatory pricing is used. This mechanism has been outlined in Section 2.2

The second form of non-for-profit attack is to sabotage MarketNet itself. For example, an agent may start destroying MarketNet money passing through a given device. Since the volume of transactions is proportional to the amount of currency transferred between client and resource managers, it is apparent that money destruction would severely reduce the utilization of the system. An alternative way to reduce system utilization would is to overload MarketNet processes such as the bank server, by issuing ever-increasing requests for currency exchanges. This difficult problem needs further investigation. Two issues should be pointed out with respect to such attacks in MarketNet: (1) malicious destruction of information is an issue orthogonal to MarketNet. For example an eavesdropper can easily alter or drop information in transit independent of *what* this information might be; and (2) the liability established through currency identifiers significantly alleviates the problem, first by identifying the source of attack and second by providing the means to isolate it.

4. Implementation issues

4.1 Implementation of Basic Components

The natural tradeoff in the implementation of components for the MarketNet infrastructure is safety vs. efficiency. Safety is important for the reasons outlined in the previous sections. Efficiency cannot be underestimated since protection, verification, and accounting may be in the critical path of resource accesses in MarketNet.

MarketNet is an ongoing project and many of the modules are under development. Due to space limitations, this section only outlines the protocol used to pay for transactions in MarketNet and briefly describes the prototype implementation that has been developed for testing purposes.

The protocol used to pay for resources and services must guarantee that no money may be counterfeit, stolen, or duplicated (for a description of the requirements for money protocols see [2] and [8]). The protocol designed for secure financial transactions in MarketNet significantly differs from typical electronic cash (*ecash)* protocols (primarily intended for payments for goods similar to those traded in real-life economies). Both the volume of transactions and the time scales at which the MarketNet protocols are required to operate on, along with the wide range in the value of the traded resources and the real-time nature of purchases, disqualify the use of ecash protocols. Specifically, the following requirements become critical in the design: (1) the storage requirements for payment related information have to be minimal; (2) the payment functionality does not significantly delay the delivery of resources or services; (3) the bandwidth requirements of transmission of payment related information are minimal.

The protocol developed for experimentation in the context of MarketNet relies in the notion of a *money string* – a random string of bits that represent money. The value of each bit is simply the amount represented by the money string divided by the number of bits in the string. One of the main ideas in the protocol is that payment is broken up to small installments along the duration of the transaction. The first payment packet contains, say, bits 0 to 63 of the money string, the second packet contains bits 64 to 127 of the money string, and so on. Bits $64*k$ up to $64*k+63$, are only valuable to the recipient as part of the

money string sequence, i.e., if the recipient also knows bits 0 to 64*k-1 of the money string sequence. This property is exploited to achieve efficiency. The first packet of the payment sequence is encrypted (encryption/decryption is in general an expensive operation) in such a way that only the intended recipient can retrieve the information contained in it. The rest of the packets in the payment sequence are sent unencrypted. Even if an eavesdropper gains access to the contents of the remaining packets, they are meaningless without the contents of the first payment packet.

The money string is first acquired from the bank, which signs the string along with other transaction-related information, with its own private key. The same money is then passed by the client to the service provider in multiple packets.

It is very unlikely that some agent may generate or steal money that can be used for profit because of the encryption of the initial money string. Further, the protocol creates incentives for both costumer and provider to behave properly. If the provider stops providing the service, the costumer will stop sending the money string. Similarly, if the costumer stops paying, the provider will discontinue the service. This constitutes a decentralized policing mechanism where without the need for third party intervention clients are assured the get the service they pay for and providers are assured the are paid for the service they provide.

The efficiency of this protocol stems from two main facts: (1) only the first packet in the stream is encrypted, and (2) verification of payment sequences with the bank is off-line. Notice that each piece of the sequence can be verified by the service provider with the bank for maximum guarantee against fraud. Nonetheless, the protocol can become very inefficient in this case. The protocol is flexible and leaves the decision of verification frequency to the provider depending on parameters such as the level of trust between the involved entities, the amount of money involved, etc.

One of the potential problems with this protocol is that an intruder may launch multiple non-for-profit attacks. One such attack could be faking a random intermediate money sequence and sending it to the provider. The latter discards it and disrupts service. One solution is to extend the protocol to include verification sequences embedded in subsequent packets. We are currently investigating these extensions.

The first prototype implementation of these components uses the C language. The architecture of the prototype is depicted in Figure 3. The user accesses remote services through a sockets-like API called *msockets* (MarketNet sockets). A typical transaction in the prototype implementation proceeds as follows. The client issues a request to the *msocket* library to open a connection to a remote service. The *msocket* routine contacts the Financial User Agent (*FUA*) module that acts on behalf of the user and asks for the appropriate currency to pay for the remote service. The *FUA* contacts the Financial Directory Service (*FDS*) to get information about the cost of the service and the currency acceptable by the service provider. It then contacts the *profile* module that stores the user budget allocation preferences to make sure that the purchase of the service is consistent with the budget allocation policy of the user. If this is the case, the bank is contacted with a request to withdraw the appropriate amount and type of currency. The bank returns the currency, after contacting the bank in the remote domain if necessary, updates the user account, and records the transaction. The FUA passes the currency to the *msocket* routine, which establishes the connection and pays for the transaction using the payment protocol described above.

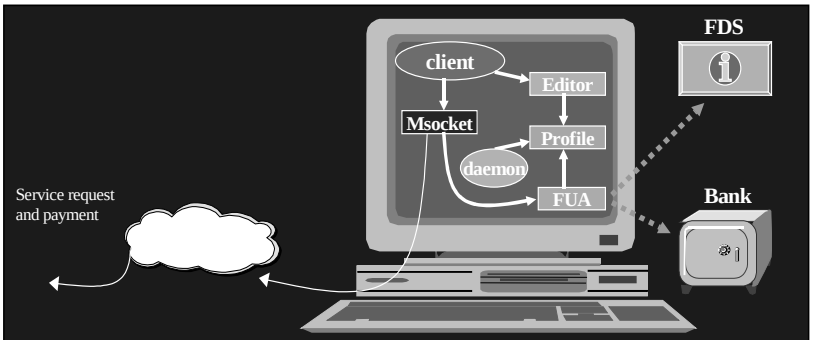


Figure 3: Prototype Implementation

4.2 Currency Issues

Currency in MarketNet may generate interesting situations that must be avoided. This section concentrates on two important ones: (1) dangers related to the power of wealth, and (2) instability of future allocations and prices.

As discussed earlier the concentration of wealth imposes a potential attack threat, since this wealth can be used to launch attacks. MarketNet develops mechanisms to eliminate this threat. First, a limit is imposed on the amount of wealth servers may keep over time. Once a pre-defined level of wealth is achieved, servers are obliged to "donate" the excess to the domain bank. Donation in this case means that the bank or some independent agency in the domain can moderate the use of the revenue. The rich agent can continue to increase its wealth. This wealth is safely stored with the bank and can be used to purchase resources. The bank will moderate its use to guarantee that it will not impose a threat on the security of the system.

There are two techniques that MarketNet can use to enforce donations. The first is to simply implement it in the server developer. This technique may be too vulnerable to mistakes or intentional misbehaviors. Another approach is to wrap the server APIs with special MarketNet APIs that can book-keep information on the server transactions. When the wealth at the server goes beyond a given threshold, the APIs will demand donation of excess wealth.

The second challenge is how to allocate resources and how to cope with large price fluctuations. For example, an agent needs to provide Web page access to multiple clients. It needs to allocate multiple resources, including bandwidth at all intermediate links. Let us assume that bandwidth is to be allocated in links L_1, L_2, and L_3. The provider acquires bandwidth in link L_1 and then proceeds to buy bandwidth for L_2. At this point L_2 is overloaded and the provider cannot offer the service. What should be done with the previously purchased bandwidth at L_1? Price fluctuations complicate the problem further. For example, assume that L_2 has enough bandwidth available but the provider may not afford it at the offered price. Finally, prices may change from the time they are advertised to the time they are purchased.

This difficult problem is being investigated in the context of MarketNet. We sketch one of the main approaches under investigation. Resources can be marketed as shares of some company. The share prices fluctuate according to the rules of supply and demand. Both share availability of and price fluctuations make life difficult for consumers.

Instruments similar to those available in financial markets (e.g., forwards, options, etc.) can help to create stability in the system ([3]). Options can guarantee that the holder will be able to purchase the shares at a given date for a give price. In the previous example, the consumer would have guarantees that she can buy bandwidth at the three links for a pre-defined price. Consequently, she may operate in a more predictable and stable environment. Additionally, if the consumer consolidates services for latter selling, she may create stable prices for her products.

We are investigating how these instruments should be priced, and exactly in what context they work. For example, the classic approach to price options is the Black-Scholes formula. The context of MarketNet may require investigations of alternative pricing strategies. For example, in some instances it may be enough to guarantee resource availability for a future time at any future market price. Alternatively, it may be enough to guarantee some amount of resources at a pre-set price.

5. Bounds on attack power

Money is synonymous with attack power in MarketNet. It is important to quantify precisely what is the attack power of an agent or domain. In the case of a particular agent such as a server, its accumulated

wealth represents its power of attack. A domain policy may force donation of excesss wealth as discussed in the previous section. This section quantifies the attack power of a domain.

The attack power of a domain is the sum of two components: (1) surplus in the trade balance with other domains and (2) domain wealth. The trade balance is the difference between the in flowing money to the domain and the out flowing money to other domains. The domain wealth is the summation of these differences from the time the domain started up to the current time. Both measures are a function of the revenues that services offered by the domain can generate and the number of costumers they attract.

What can one do to bound the attack? Domain policies can enforce the bounds on trade flows and consequently limit the accumulation of wealth. For example, one may force a domain to always balance in flow and out flow. Ideally, a zero trade balance will force domains to not accumulate wealth. Another policy could be that the domain must convert excess wealth in some special currency where the transaction is logged. For example, a domain may be forced to convert revenues to dollars.

Another protection is the price mechanism in MarketNet. Attacks to resources will increase the demands for services and consequently increase prices. Eventually the domain budget finishes and the attack is contained.

Let us now consider the situation when a domain launches an attack to some other domain. It may operate according to the domain policies but still attack. It may change its patterns of expenditure so that it balances its trade, but the out flow of money is directed to attack. In this case, the power to attack is really a function of the domain wealth plus the revenues its resources may generate. This situation can only be contained issuing an embargo against the domain that will discourage or forbid other domains from acquiring resources that belong to the malicious domain. Such decisions may be issued by the central bank.

6. An Example: The Worm Attack

In this section we take as an attack example the famous “worm attack” ([9]) and show how MarketNet would have reacted to it. The purpose of the worm was to spread itself, i.e., to break into systems, install and locally run itself. To achieve this goal it exploited flaws in utility programs based on BSD-derived versions of Unix. The worm program eventually spread itself to thousands of machines around the Internet, and disrupted normal activities and Internet connectivity for many days. The disruption was caused by the worm’s heavy use of system resources in order to break to other systems. The worm was not destructive in nature, i.e., it did not delete user files or try to cause other such disruptions, but it has to be noticed that if its intention were to do so, nothing would have prevented it.

The worm spread using three techniques:

1. Exploiting a bug in the *gets* standard C I/O library used by the finger daemon (*fingerd*). The *gets* call takes input to a buffer without doing any bounds checking. The worm would overran the buffer and rewrite the stack frame, allowing it to gain control in the target machine and install and execute itself locally.
2. Using the *debug* option in *sendmail* servicing the SMTP port. This feature enables testers to run programs on remote machines to display the state of the remote mail system without sending mail or establishing a login connection.
3. Guessing passwords and exploiting trust between hosts. Once present in a system the worm would try to guess user passwords, break into their accounts and exploit trust between hosts (e.g., hosts found in /etc/hosts.equiv and /.rhosts) to remotely execute into those systems as well. Password guessing is a computationally very intensive operation using a big portion of the system resources.

Using any of the above methods, the worm would successfully spread without leaving any trace of where it came from.

MarketNet can protect against several features of worm-like attacks:

1. The prospective attacker would have to pay to use *sendmail* or *fingerd*, leaving an unforgeable trace of the originator of the attack.
2. Using system resources is not free. To perform password guessing the process would involve heavy system resource utilization. Monitoring of the budget usage at the conquered account domain would soon trigger alarms due to the unusual behavior. Furthermore, the amount of damage (e.g., overloading system resources) the process can achieve is limited by the budget available to it. Notice that we make a worst-case assumption, namely that the intruder manages to use the budget available to the account for using the system resources. Mechanisms to impose restrictions on the budget available to processes are currently under investigation in MarketNet.

MarketNet protects systems without eliminating software bugs. It assumes that software bugs are always very likely to exist and creates a layer of protection that is independent of the correctness of software.

The worm attack is one of the most difficult attacks to handle and shows some of the limitations of MarketNet. These limitations are not particular to MarketNet. The limitations under consideration stem form the fact that software implementation bugs may allow intruders to impersonate legal users of systems and therefore gain the same privileges the legal user would have. We are currently investigating how MarketNet can efficiently react in the following scenarios:

Assume that the worm had destructive intentions. Budget enforcement along with usage monitoring in MarketNet would limit the scope and the extent of the damage. We are currently investigating price discrimination techniques that may be able to limit such attacks by making resources very expensive when the process does not normally use it. For example, deletion of files would be very expensive for unknown processes, which will not have enough money for the attack.

In a worm-like attack, the attacker manages to impersonate the owner of an account. Even when this happens, it should not be equivalent to getting hold of the budget of the conquered account. One of the mechanisms to break this equivalence is usage monitoring. Abnormal access patterns can be restricted providing adjustable limits on the amount of damage a malicious or faulty processes can cause. A second mechanism under investigation is the separation of budgets available on a per process and/or per task basis. The tradeoff in this case is protection level vs. ease of use of the system.

7. Conclusions

Market-based technologies can provide an effective solution to the protection of information systems. MarketNet develops unified, systematic market-based technologies to provide scalable and tunable access control for large multi-domain networked systems. These technologies enable unified monitoring and correlated analysis of resource access to detect intrusion attacks, isolate the sources of attacks and respond quickly to control its damages. MarketNet develops mechanisms to protect critical network services, based on their quantifiable value to users, and assure their continuous availability through failures or attacks based on user's priority.

In summary, some of the key ideas in MarketNet are the following:

- Currency is used to provide unified, scaleable, resource-independent access control to resources and services and account for their use.

- Resources and clients are organized in currency domains. Each domain has its own currency. Clients wishing to access a resource must pay in currency acceptable to the domain that owns the resource. A domain has full control over its exposure to attacks, by controlling access to its resources through several parameters: the price of a resource; the budget allocated to a given client; and the rate at which currency is provided to a given client.
- Organization in currency domains can limit the spread of faults and attacks.
- The power of attacks is limited by the budget available to the attacker and by the price of resources.
- Currency carries unique unforgeable identifiers that can be monitored and traced back to the holder. Currency identifiers establish verifiable accountability on the use of resources.
- Currency provides a resource-independent instrumentation to monitor and correlate access patterns and to detect intrusion attacks through automated, uniform statistical analysis of anomalous currency flows.
- Prices are dynamic. They can be used to fine tune access control to resources. They provide the means for optimized load redistribution and graceful degradation upon loss, and impose quantifiable dynamically adjustable limits on the exposure to attackers.

These mechanisms are resource-independent, and admit unlimited scalability for very large systems consisting of federated domains operated by mutually distrustful administrations

Bibliography

[1] Clearwater, S., editor. "*Market-based Control of Distributed Systems*," World Scientific Press, 1996.

[2] Dailianas, A., and Y. Yemini "*A Protocol for Secure Financial Transactions*," Paper in Preparation.

[3] Hull, J. C. "Options, Futures, and Other Derivatives," third edition, Prentice Hall.

[4] Kurose, J., M. Schwartz, and Y. Yemini "*A Microeconomic Approach to Optimization of Channel Access Policies in Multiaccess Networks*," Proc. Of the 5th International Conference on Distributed Computer Systems, Denver, Colorado, 1995.

[5] MacKie-Mason, J., and H. Varian "*Economic FAQs About the Internet*," in The Journal of Economic Perspectives, vol. 8, no. 3, pp. 75-96, 1994. Reprinted (with revisions) in Internet Economics, J. Bailey and L. McKnight, eds. Cambridge, MIT Press, 1996.

[6] MacKie-Mason, J., and H. Varian "*Pricing the Internet*," in B. Kahin and J. Keller, editors, Public Access to the Internet, ACM, Boston, Massachusetts, May 1993.

[7] Sairamesh, J., D. Ferguson, and Y. Yemini "*An Approach to Pricing, Optimal Allocation and Quality of Service Provisioning in High-speed Packet Networks*," in Proc. of the Conference on Computer Communications, Boston, Massachusetts, April 1995.

[8] Schneier, B. "*Applied Cryptography*," second edition, John Wiley & Sons, pp. 139-147.

[9] Spafford, E. "*The Internet Worm Incident*," Technical Report CSD-TR-933, Department of Computer Sciences, Purdue University, Sept. 19, 1991.

[10] Walsh, W., M. Wellman, P. Wurman, and J. MacKie-Mason "*Some Economics of Market-Based Distributed Scheduling*," In Proc. of the 8th International Conference on Distributed Computing Systems (ICDCS-98), Amsterdam, the Netherlands, May 1998.

[11] Yemini, Y. "*Selfish Optimization in Computer Networks*," Proc. of the 20th IEEE Conference on Decision and Control, pp. 281-285, San Diego, CA., Dec. 1981.