Fluid Information Systems

Christian W. Probst
Technical University of Denmark
probst@imm.dtu.dk

René Rydhof Hansen
Aalborg University
rrh@cs.aau.dk

ABSTRACT

Networked communication systems and the data they make available have, over the last decades, made their way to the very core of both society and business. Not only do they support everyday life and day-to-day operations, in many cases they enable them in the first place, and often are among the most valuable assets. The flexibility that makes them so valuable in the first place, is also their primary vulnerability: via the network, an entity's data is accessible from almost everywhere, often without the need of physical presence in the entity's perimeter. In this work we propose a new security paradigm, that aims at using the network's flexibility to move data and applications away from potential attackers. We also present a possible realization of the proposed paradigm, based on recent advances in language-based security and static analysis, where data and applications are partitioned ahead-of-time and can be moved automatically based on activity both in the network as well as the real world.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems; D.4.6 [**Operating Systems**]: Security and Protection—*Access controls*; H.3.4 [**Information Storage and Retrieval**]: Systems and Software

General Terms

Security, Design

1. INTRODUCTION

At the very core of businesses and society, information systems pervade our daily life. They are essential at all levels of actions, from meaningless "surfing" to decisions that potentially can influence the well-being of companies, economies, or even whole populations.

Not only do these information systems support everyday life and day-to-day operations, in many cases they enable them in the first place, and often are among the most valuable assets. Offering seamless access to computing resources and data from virtually any location around the globe is one of the most outstanding features of these systems. This flexibility makes them so valuable, but at the same time is the reason for their major vulnerability: via the network, an entity's data is accessible from almost everywhere, often without the need of physical presence in the entity's perimeter. With upcoming techniques such as cloud computing, owners of data and applications are not even able to physically enforce access control to their assets any longer.

This risk of data possibly being accessible without proper authorization has lead to a wide range of access control mechanisms, which are supposed to restrict access to data. Protection usually consists of an onion-like layering of firewalls, encryption, and other kinds of access control mechanisms. However, with every combination of protections like these, eventually the layers are breached and data is leaked.

While we do not claim that the paradigm suggested in this paper is insurmountable, it is to our knowledge the first approach to moving the data away from potentially insecure locations. The new security paradigm proposed in this work aims at using the network's flexibility to move data and applications away from potential attackers. The paradigm is based on our belief that in order to counter threats to mission-critical data we need an infrastructure that makes it possible to dynamically distribute and move data and applications.

Overall our approach is closely related to risk analysis—an organization will need to assess how much risk it is willing to take as to making (possibly protected) data accessible to anybody with access to the location where it is stored. Even worse, this does not only hold for *data*, but also for *applications*. If one has access to a host that runs (parts of) an application that deals with sensitive data, then one should either be trusted not to try to get access to that application's memory slice, or the application should either be stopped or relocated and the memory slice be erased. This is exactly what the suggested paradigm is designed to address.

We also present a possible realization of the proposed paradigm, based on recent advances in language-based security and static analysis, where data and applications are partitioned ahead-of-time and can be moved automatically based on activity both in the network as well as the real world.

The partitioning is based on programming languages such as Jif that support the annotation of data and code in an application with owners and admitted accessors [15]. The usual targets of Jif are applications with mutually distrusting owners of data and applications, who want to ensure that unauthorized accesses to their assets are disallowed by the underlying system. The program is executed on a network of hosts operated by the principals, and the runtime system ensures by construction that the distribution of the application onto the network obeys all access-control annotations of the

NSPW'09, September 8–11, 2009, Oxford, United Kingdom

original code. In previous work [11, 23] this approach has been adapted to deal with dynamically changing networks, where both the network and/or the group of principals can change.

In the suggested paradigm we use static analysis techniques [17] to guide the computation of partitions. We use the information on principals in the system along with their access rights to code and data to compute possible partitionings ahead-of-time. An analysis similar to the one presented in [21] allows us to pre-compute which principals can reach which location, and which data should or should not be accessible. Note that in the grid, users and hosts may change dynamically (and frequently) and so the static partitioning must be based, e.g., on roles or the organization to which an actor belongs.

Combining these two approaches allows the system to automatically move data and code away from both physical and virtual locations based on physical and virtual actors. Since the hosts in the network as well as the actors are known ahead-of-time, possible scenarios can be pre-computed as well, and thus more effectively disallowing access to data if the actor performing the access does not have the necessary rights.

Compare this to the tactics of guerrilla armies. Guerrilla warfare usually is characterized by a considerably sized army, that when under attack splits up in small, independent units. These units continue to follow the original goal, partly independent, partly collaborating with other units. What makes these tactics so successful is the high flexibility, and the ease of splitting and re-uniting units.

The overall goal of both guerrilla tactics and the suggested paradigm can be compared to the experiment of nailing a pudding to a wall [7]—whenever one tries to nail the pudding (access the code or data), it slips away (is relocated) to another host. Another analogy are magnetic poles pushing equal poles away. This approach has supposedly been used quite successfully by Scrooge McDuck to move his vault away from the Beagle Boys.

The rest of this paper is organized as follows. In the rest of this section we lay out two scenarios where our paradigm would be able to provide better protection of data. In the next section we present the paradigm and its constituents, followed in Section 3 by a more detailed description of the paradigm's constituents, and a discussion of how to strengthen (or weaken) the approach, as well as of approaches for enforcing its success. Section 4 and Section 5 discuss drawbacks and benefits in a system based on the suggested paradigm and implementation issues, respectively. Section 6 briefly discusses related work, followed by a conclusion and an outlook on future work in Section 7.

1.1 Example Scenarios

Before introducing the paradigm of fluid information systems, we briefly discuss some example cases and how they could benefit from the paradigm. The setting of our example scenarios are a bank, representing a traditional, administrative domain, and a grid-computing service with remotely managed resources.

A banking system.

Consider a bank department that deals with loans and credit reports. To protect the bank against malicious actions by an inside actor, each document and contract must be signed by (at least) two people. In this "traditional" setting, the document in question can either first be signed by one actor, and then by the other, or it can be signed by the same actor, using two different signatures.

Assuming a network-based document-storage system, sensitive data will (again in the traditional setting) be available on the host(s) where it is stored, independent of whether an actor accessing the system needs to have or has access to the data or not. This means that with every actor having access to the location but not to the data, the risk of the data being accessed illicitly may increase. Therefore, sensitive data should be re-located whenever a user with insufficient rights enters or gets access to the location. This scenario is gaining special importance with the spreading of technologies such as cloud computing, where the data owner has little, if any control over where data is stored.

The cases discussed so far rely on internal events that are potentially easy to detect by hooking up a door with a badge reader to a logging service or checking computer logs, thus detecting, e.g., robots entering the system or users entering a room or logging into a system. Beyond these cases, there will also be events that may not always be detectable. For example, consider a spyware or virus scanner, that triggers an event in the case of detecting an intrusion. While quarantining the piece of software in question is a viable option on user machines, this may be unacceptable on servers, where it would be preferable to simply move any potentially threatened data.

A grid-computing service.

In the previous scenario, the bank has complete overall control of the IT-system. Moving to a cloud or grid based system this is no longer the case. Indeed, the system underlying the cloud (or grid) may be owned by one or more entities completely different from the users of the copmuting services provided by the cloud. This makes it very challenging to provide security for computations taking place in the cloud and even more challenging to guarantee relevant security properties.

As an example, consider a biotech company that outsources computationally complex tasks to a grid, *e.g.*, analysis of genomes. Typically the data produced by such analysis is highly sensitive and must be kept confidential[1]. Obviously the biotech company wants some assurance that its data and computations are adequately protected. However, since data and computations can be moved between nodes in the grid dynamically, *e.g.*, for better load balancing or to compensate for failing nodes, it is impossible to predict where specific data items or computations end up and in particular what other data and computations a given node is shared with. The dynamic nature of this scenario makes it difficult to use and apply traditional security measures and policies in a meaningfiul way.

2. FLUID INFORMATION SYSTEMS

Before presenting more details, we state the paradigm proposed by this work. We believe that in order to counter threats to mission-critical data and applications in networked systems, we need an infrastructure that makes it possible to dynamically distribute and move data and applications on hosts in the network:

> Instead of simply protecting data and applications by static and stationary defense mechanisms, e.g., firewalls and access control, with an (often implicit) underlying assumption of a fixed and relatively stable infrastructure, resulting in a security paradigm not unlike a digital Maginot Line, we need a paradigm-shift towards systems that protect data and applications through decentralization, decomposition, and high mobility of data and resources.

It should be noted that this is not supposed to replace our current protection mechanisms, but to enhance them with a mechanism to move the data and applications away once they are threatened.

[1]This sensitivity of data, which, in principle, is incommensurate with the idea of cloud computing, is exactly one of the threats to the idea of cloud computing, but that is a completely different story.

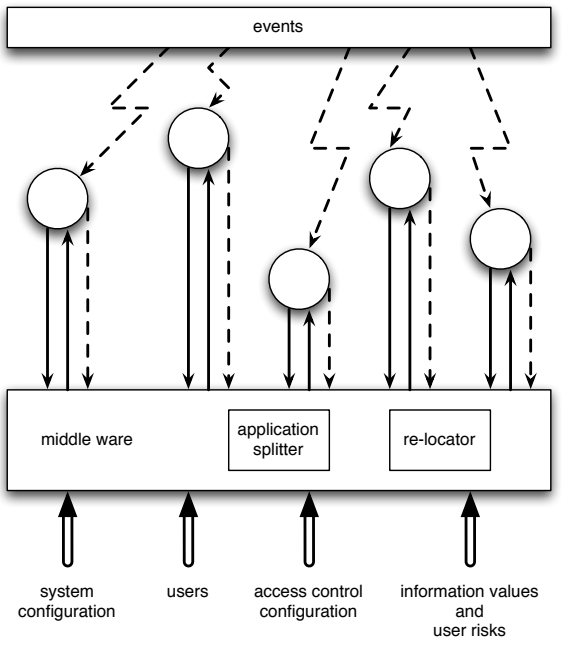

Figure 1: Elements in an instance of the suggested paradigm. The circles represent nodes in the physical or virtual domain, where data and applications are stored and executed. Whenever events occur they are forwarded to the middle ware component, which then decides, based on the configuration components shown on the bottom of the figure, whether or not data and application parts should be re-located.

The idea is that, based on access-control mechanisms, both data and applications should only be available on those network nodes where *only* users are present who may access them, and even more importantly *no* users are present, who may *not* access them. To this end, data and applications should be moved away from nodes that do not fulfill these properties, thus also moving (or being moved) away from possible attackers. This can be compared to trying to nail a pudding to the wall [7]—whenever the attacker tries to nail the data or application, it already has vanished.

It is important to note that the idea is to distribute the application and data, and keep them distributed. This means that during actual operation there is no need to re-unite all bits and pieces, but instead the application is executed in a distributed manner, ensuring correct access to other parts of the application and data.

One shortcoming of the paradigm as just described is that it is too "black and white". Assume the bank scenario as described above. If Alice needs to access a document, Bob can hinder her from doing so simply by logging into the system, or entering the room. If the system did not have any shades of gray, Alice would never be able to access the document as long as Bob is in the room or logged in. This goes back to the brief mention of risk analysis in the Introduction. In principle each piece of information should be associated with a value to the organization, and each actor with a risk value. Information's value expresses the information's importance to the organization, and in the end determines how big a risk the organization is willing to take with respect to this information. Together with the risk value(s) of the actor(s) being present at the location where the data is stored, the suggested paradigm allows the system to determine whether or not the data should be re-located.

2.1 Elements of the Paradigm

In order to implement a system living up to the goals of the suggested paradigm, it needs to support certain operations and features. Before presenting a concrete instantiation in the next section, we briefly discuss the paradigm's building blocks. The overall structure is shown in Figure 1.

- **System Configuration.** The elements shown at the bottom of Figure 1 constitute the core information on the system. They determine the structure of the underlying infrastructure (both physical and virtual), the users in the system and the access control specification, as well as the risk associated with individual users and the value of information items.

- **Actors and Events.** These elements are tightly coupled—actors have certain access rights on data (and applications), and they cause events, that can have many different sources. For example, actors might log into systems, enter locations, access data, etc. In principle, the kind of events, which can trigger a system response, is only limited by the support of the environment. As discussed above, they might also include triggers from spyware software or virus checkers, include proximity sensors, etc.

- **Data.** We subsume both information and applications under data, since they should be dealt with uniformly. Data can be re-located to other hosts, and, in principle, could also be split across hosts if different parts had different access restrictions. Applications should also be re-locatable, and can be split into parts dealing with certain groups of data that has similar restrictions and owners.

- **Middle ware.** The middle ware is the building block in realizing the paradigm. It ensures that data and applications are split and re-located based on the events triggered by actors in the system.

2.2 Example Scenarios Revisited

A banking system.

The banking scenario presents a straightforward application of Fluid Information Systems: here sensitive data is simply moved away from network nodes accessed by non-trusted actors, be it users or automated administrative processes such as malware scanners.

A grid-computing service.

For grid based services, Fluid Information Systems may offer an approach to security that is more directly applicable and easily related to the underlying grid structure. The notion of making data and computations more decentralized and mobile seems to be a very good fit to the paradigm of grid-computing.

In the case of the biotech company performing sensitive copmutations on confidential data, Fluid Information Systems enables the company to specify a security policy, *e.g.*, guaranteeing that sensitive computations are only performed on nodes that are not shared with other, potentially malicious, agents. This approach may be taken even further by requiring that during an attack the most sensitive computations must be split into smaller, and presumably less sensitive, computations that are then scattered across the entire grid. Once the attack is over, the computations may gather and aggregate again into fewer but larger computations that make better use of the provided resources.

To prevent "shepherding attacks" (see below) and similar attacks to drive data into specific parts of the grid or to isolate computations at particular nodes, it would be possible to specify that particularly sensitive data or computations must *always* be moving between randomly chosen nodes. This would make it very hard for an attacker to locate and attack individual computations.

3. AN INSTANCE OF THE PARADIGM

In this section we present in more detail the constituents of an instance of the system suggested by the paradigm. It uses program partitioning as introduced by Myers *et al.* [27, 28] to realize the re-partitioning and re-location of applications, and insider threat analyses as introduced by Probst *et al.* [21, 19] to identify which locations can be reached by which user. It should be noted that all computations of reachability and partitions can be performed off-line, thereby reducing the run-time overhead. At run-time, the middle ware is only concerned with catching events, and uses these events to re-locate data and applications. The description in this section is closely based on the techniques presented in [21] and [23].

3.1 Program Partitioning

Information-flow policies have been used for specifying confidentiality and integrity requirements. Their success is mostly based on the ability to specify how information may be used in the system as opposed to which principals may access or modify the data. Security-typed languages [14, 22] have been used to implement these policies in programming languages. By annotating data in programs, the programmer explicitly specifies how the flow of information allowed by the language semantics should be constrained. The benefit of these explicit annotations is that programs that violate the restrictions will be rejected either during compilation or during execution. Thus, the program itself does not have to be trusted—instead, only the reused components (compiler and run-time system) must be trusted.

Secure Program Partitioning [27, 28] is a language-based technique for distributing confidential data and computation across a distributed system of mutually untrusted hosts. The program to be distributed is annotated with security types that constrain permissible information flow as described above. The resulting confidentiality and integrity policies are used to guide the partitioning of application and data across the network. The resulting communicating sub-programs not only implement the original program, but at the same time satisfy all security requirements of principals, including trust relations to other principals as well as hosts. The results reported in [27, 28] with respect to the performance of the distributed code suggest that this is a feasible way to obtain secure distributed computation.

The main drawback in Secure Program Partitioning (SPP) is that the framework contains two static components—the *trust relation* and the *hosts in the network*. While a static trust relation is necessary to ensure system integrity, the second restriction not only seems to be superfluous, but also hinders the system from moving data or computations to newly available hosts that might allow a better fulfillment of a principal's requirements. Thus, by allowing the partitioning to be adjusted after a host joins or leaves the network, the overall trust of the principals in the partitioning—and thus in the security of the distributed system—can be increased.

In an ideal world, hosts should not store any references to items that have been moved to another host after data and computations have been redistributed in the enhanced network. The recently introduced mechanism of Erasure Policies [6] allows one to design systems where programmers annotate their data with policies that describe exactly this kind of behavior. Erasure Policies state explicit erasure and declassification requirements, and in our framework would be enforced by each client's portion of the framework.

While [11] and [23] are based on dynamically changing networks and sets of principals, in realizing the paradigm we turn the tables—in an information system we know statically which nodes are available, and who operates them, and we know statically which actors or access rights are assigned in the system. The exception to this are nodes that are mobile such as laptops, or nodes that are only available intermittently.

The benefit of SPP is that participants do not need to fully trust each others' hosts to enable the distributed execution of a program dealing with their data. As Figure 2 depicts, in the original framework introduced by Myers *et al.* in [27], the compiler receives two inputs: the program source code, which uses a security-typed language, and the trust declarations of all participants. These declarations state each principal's trust in hosts and other principals. They are used to guide the compilation and splitting of the security-typed program into sub-programs that are executed on (some of) the hosts in the network. By communicating, these sub-programs perform the same computation as the original program, however, the splitter ensures that all trust and security policies are fulfilled. As stated by Myers *et al.* [27], the splitter ensures that if a host h is subverted, only the confidentiality or integrity of data owned by principals that trust h is threatened.

The main component in the programming model used in SPP is the *principal*, who can express confidentiality or integrity concerns with respect to data. Principals can be named in information-flow policies and also define the authority possessed by the program being executed. Security labels [15] express confidentiality policies on data. A label $l_1 = \{o : r_1, r_2, \cdots, r_n\}$ means that data labeled with l_1 is owned by a principal o and that o permits readers $r_1, \cdots, r_n$ (and o) to read the data. Data can also have multiple owners, each expressing its concerns with respect to the data. E.g., $l_2 = \{o_1 : r_1, r_2; o_2 : r_2, r_3;\}$ expresses that owner o_1 allows readers r_1 and r_2 to read data labeled with l_2, and owner o_2 does so for readers r_2 and r_3. Of course each annotation must be obeyed by the system, that is only r_2 will be allowed to access data labeled with l_2. Additionally, labels may specify integrity. $l_3 = \{? : p_1, \cdots, p_n\}$ specifies which principals trust the data labeled with l_3.

3.1.1 Application to the Paradigm

In the scenario targeted by the paradigm suggested here, we face a different situation. We know precisely which nodes are available in the network, namely which hosts are trusted to store data and/or execute code. This is an important distinction to the general setting described above, since it allows one to make much stronger assumptions as to where to put data and code. Furthermore, since all this information is known beforehand and statically, the partition-

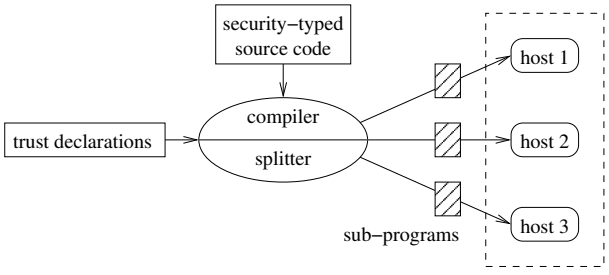

Figure 2: Structure of the Secure Program Partitioning Framework.

ings described above can be pre-computed for all or at least many possible scenarios as to which user might be at which location (or execute a program at which location).

For obvious reasons, the number of partitionings to compute can easily become intractable, due to large numbers of principals, hosts, and data in any information system of reasonable size. In order to actually make realizations of the suggested paradigm feasible, we suggest combining program partitioning with a static-analysis technique such as the technique suggested in [21] that effectively allows the system to pre-compute which actor can access which locations and which data in a network.

3.2 Threat Analysis

In previous work [21, 19, 20] we have developed a static-analysis technique based on a formal model of a system. This technique allows us to analyze the system model for potential insider threats [3]. Most often the primary measure taken to protect against insider attacks is still *a posteriori* auditing of log files when an insider incident has been detected [4].

In instantiating the paradigm, we are in a situation where we have a model of the system as described above, and have the configuration of access control available. This enables application of techniques such as the ones described in [20, 21], where a static analysis is used to pre-determine which actors might reach which locations, and which data might be accessed. Together with the system's access control configuration, this analysis can be used to determine when data and code must be re-located to which other nodes. Our previous analyses [20, 21] are based on a model that allows one to define a notion of insider attacks, and thereby to study systems and analyze the potential consequences of such an attack. Formal modeling and analysis is increasingly important in a modern computing environment with widely distributed systems, computing grids, and service-oriented architectures, where the line between insider and outsider is more blurred than ever—which is especially true for the kind of information systems targeted by the suggested paradigm.

The formal model [19] also suggested for this work has two parts: an abstract high-level system model based on graphs, and a process calculus, called *acKlaim*, providing formal semantics for the abstract model. As the name suggests, the acKlaim calculus belongs to the Klaim family of process calculi [16] that are all designed around the tuple-space paradigm, making them ideally suited for modeling distributed systems like the interact/cooperate and service-oriented architectures. Specifically, acKlaim is an extension of the μKlaim calculus with access-control primitives. In addition to this formal model we also show how techniques from static program analysis can be applied to automatically compute a sound estimate, i.e., an over-approximation, of the potential consequences of an insider attack. This result has two immediate applications—on the one hand it can be used in designing access controls and assigning security clearances in such a way as to minimize the damage an insider can do. On the other hand, it can direct the auditing process after an attack has occurred, by identifying *before* the incident which events should be monitored. This flexibility is exactly what makes it so suitable for the application suggested here.

An important property of the suggested model is that it enables the formalization of many *soft aspects* of the system. The degree of abstraction in the system model is sufficiently high to allow easy modeling of real-world systems, yet low enough that technical details relevant to security can also be modeled with relative ease. The acKlaim process calculus is used to formalize the semantics of the system model itself and can be used to formally reason about properties of the system. The system model is based on a system consisting of locations and actors. While locations are static, actors can move around in the system. To support these movements, locations can be connected by directed edges, which define freedoms of movements of actors.

3.2.1 Application to the Paradigm

As described above, the analysis just described is ideally suited to be applied in the setting of the suggested paradigm. It effectively allows one to model the information system, and thus forms the basis for limiting the number of potential partitionings to be computed by the program partitioning framework.

At the same time, the result of the analysis can be used to identify shortcomings in the overall system setup. Obviously, one of the prerequisites for realizing the paradigm is the ability to sense the arrival, presence, and departure of an actor at a location, since this capability is essential for starting re-location of data and code, if necessary. This relates closely to, e.g., work on data tethers [2], which aim at enabling the automatic protection of data based on environmental circumstances.

3.3 Strengthening the Paradigm

The core of the suggested paradigm is to ensure that data and code are not accessible at a node if they could be accessed by an actor who is present at that node (or can access it) but should not have access to this very data or code. To make the paradigm acceptable, a proof of this denial of success seems to be necessary—and is exactly what can be achieved by means of techniques such as the recently introduced mechanism of Erasure Policies [6]. Erasure policies allow to design systems where programmers annotate their data with policies that describe exactly this kind of behavior. Erasure Policies state explicit erasure and declassification requirements, and formal methods can then prove that these annotations are actually obeyed. Applying Erasure Policies means that one can formally prove, once and for all, that a framework is strong enough to uphold the paradigm, for example, by erasing all code and data from, e.g., a node's memory whenever the code and data have been repartitioned to another node.

As mentioned above, data tethers [2] can be used to further strengthen the suggested paradigm, by increasing the possible interactions between the system and its environment. Examples might include disallowing certain documents to be stored on the same host at the same time. Data tethers would also allow to realize policies at a higher level, such as, e.g., the four-eyes principle, where data may only be accessed by two authorized persons simultaneously. In the setting of the paradigm this would mean that once the first actor enters a location, the framework ensures that the data to be accessed is partitioned to another host. Once a second actor enters the location, and if and only if this actor has sufficient rights to allow access to the document, the data is scheduled back to the location. Note that data tethers can be realized by many different means, e.g., logging mechanisms.

Another approach for strengthening the paradigm is to make data and code disappear in situations that would be considered especially dangerous. This could be combined with approaches such as [12], which aims at securing critical data in hard-to-attack regions of a system until it is safe to make the data available more freely again. Clearly, also this extension would benefit from data tethers to allow more fine-grained sensing of both the beginning of a dangerous situation as well as its dissolving.

3.4 Weakening the Paradigm

One of the strengths of the paradigm described above is its ability to reason about the underlying system at a very detailed level. This

ability allows one to model and capture even smallest nuances of the critical aspects of the system under consideration, but at the same time can result in an overwhelming number of little details.

While this is a typical risk with static analysis techniques, they also provide a solution—the selection of more coarse-grained abstractions that allow to treat bigger systems in less time. Once the (less precise) result has been obtained, interesting system parts can be chosen to be re-analyzed with a more fine-grained abstraction. Now, since the sub-system in question is smaller than the whole system, the analysis time will be shortened.

3.5 Enforcing Paradigm Success

Beyond the requirements discussed above, there is one major obstacle to overcome, which is the challenge of enforcing the paradigm's success. Often, new approaches are challenged to argue for how they can ensure to be able to operate under attack. For obvious reason, this would be an important feature of a new paradigm or technology. On the other hand, the authors think that this property might be highly overrated, since also for existing technologies it often is unclear as to how they ensure such a property. If, for example, the cable to the red phone is disconnected, then it will be hard to argue that this is a failure per se of the idea of the red phone [26].

For this reason we believe that the only really important condition for the presented paradigm's success (as well as the success of other approaches) is that the infrastructure described above is in place and operational. While many direct threats to the system itself will not be preventable, we think that the benefits gained outweigh this. Even more importantly this property is very similar to that of real systems as described above, so at least it does not represent a worsening of the status quo. In the next section we briefly discuss some of these issues.

4. SECURITY ISSUES

In this section we discuss briefly some of the risks faced in a system implementing the suggested paradigm.

Trapping data.

An important question is how to prevent data from being "trapped", or access being denied by a not-authorized user being present in a location (physically or being logged in) where an authorized user wants to access the information. As discussed above, this can easily happen in a "black and white" instantiation of the paradigm, which can not distinguish between different levels of risk posed by users, or value of information. Therefore, in principle, each piece of information should be associated with a value to the organization, and each actor with a risk value. Information's value expresses the information's importance to the organization, and in the end determines how big a risk the organization is willing to take with respect to this information. Together with the risk value(s) of the actor(s) being present at the location where the data is stored, the suggested paradigm allows the system to determine whether or not the data should be re-located.

Beyond this, users should also be associated with the trust the organization has into them. If a user is very trustworthy it may very well be acceptable that he accesses data even though non-trustworthy actors are present and potentially could access the data.

Shepherding Attack.

With this new paradigm also come new attack strategies. One strategy in particular must be dealt with by any instance of the paradigm, what we call the "Shepherding Attack". The Shepherding Attack is an attempt at driving data into "dead ends", i.e., places where data is of little or no use or maybe not even accessible to legitimate users of a given system resulting in a denial-of-service attack. Alternatively an attacker may try and drive data into locations that are controlled by the attacker or at least more susceptible to other means of attack.

Preventing shepherding attacks in practice depends on a number of things including the specific nature of the data to be protected, what the data has to be protected against, and the underlying network. For some applications, e.g., the banking example described in the introduction, it would make sense for sensitive data to be moved into a highly secure storage node whenever a non-trusted user is logged into the system. In contrast, in the grid example also discussed earlier, it might be better to split a computation into several smaller computations, and possibly replicate some of them, and then distribute the smaller computations even further in the grid thereby making it harder to prevent the overall computation to finish. As these examples show, the security requirements and strategies imposed by different instances of Fluid Information Systems can vary greatly. Therefore every specific instance of a Fluid Information System must also include a security policy, based on rigorous threat analysis and risk assessment, that specifies how the system should act in during an attack, e.g., a Shepherding Attack.

Threat Model and Events.

The threat model clearly depends on the scenario where the paradigm is being applied, and on the organization's general risk assessment. Depending on this assessment, the risks to consider can for example, when considering accessibility of the local infrastructure, range from purely local threats to remote threats; when considering the execution environment they may include threats due to data made available on the cloud. The important point is that there is no universal threat model built-into the paradigm, but that the paradigm can be adapted to the current situation.

The same is true for the types of events to deal with, which could range from simple access-control or IDS events to major disaster alerts such as terrorist attacks.

In both cases the paradigm can be used to realize a flexible response, meaning that for events and threats classified as being "ordinary" the application and data are re-distributed as described above; this can be imagined to be the unauthorized user entering a room or logging into a work station. In the case of an extraordinary event or threat, the application might react completely different, for example moving data and applications to a safe area, which subsequently is disconnected from the network.

The Cloud.

With respect to the suggested paradigm, cloud computing or "the cloud" can play different roles. A cloud infrastructure could be the obvious target place for moving (parts of) applications and data, potentially making it available to a wide selection of nodes in the organization's network at the same time. This would significantly ease the burden on the splitter and the mover. On the other hand, the cloud also complicates ensuring the paradigm's success, since the ability to observe actions of actors on remote machines relies on the cloud's operator. Similarly whether observed actions can be trusted or not relates to the trust into the cloud's operator. Therefore, the overall contribution of a cloud-computing environment depends on the specific system setup.

Availability.

A clear issue is that of availability of data. While in the "original" system, data would be stored at a single host, it now may be

distributed over a large number of machines. This makes it much more likely that part of the data can be accessed be an adversary, and it makes it much harder to protect all machines that store parts of the application and data.

Clearly there exist two attacks of major concern; obtaining a complete data item or making a data item unusable by destroying one or more parts of it. The former attack is obviously becoming less and less likely the more hosts the data is distributed across. The latter can be countered by using data redundancy approaches when performing the splitting.

When comparing a system implementing the paradigm to a system *not* implementing it, it should be noted that the chance of an attacker to either obtain the complete data item or make it unusable is significantly lower in the paradigm system because of the distribution of data. This could enable system operators to give protection guarantees for data and applications.

5. SETUP AND ADMINISTRATION

While there are often numerous challenges to overcome when implementing new security technology, not least when this represents a change in paradigm, two areas seem to be of special concern to a successful implementation:

- Movement of data. The paradigm by nature relies heavily on being able to move substantial amounts of data between network locations, in order to keep the data secure. Under certain circumstances it may not be possible to actually move all the sensitive data in time, e.g., if all network connections are down or because the amount of data simply is too big.
- Administration. Clearly, administering a system where data may not be where the administrators expects it to be, or indeed where data may be moved without notice, may present even experienced system administrators with an interesting challenge.

Data.

One implementation strategy for overcoming the problem of data movement is to encrypt, partition, and replicate all data that is to be protected. This may help solve the data movement problem in several ways: by requiring that all data must be encrypted it is now possible to "move" data simply by first moving the decryption key. In a more elaborate scheme data could be encrypted and forwarded to potential next locations; once the data needs to be moved initially only the key is moved, and the data and the previous location can then be deleted. While this poses new problems with respect to synchronization, this is a fairly well understood problem in databases. Further, by partitioning and replicating data this scheme may also allow to move a relatively small partition rather than a full data set.

Another approach to deal with especially very large data sets is to make certain parts of the system immobile. Consequently they will always be scheduled to the same node, and will thus not be moved around.

Administration.

Solving the problem of administration may require a more radical approach, e.g., a "friendly" worm designed to perform administrative tasks. On the other hand, since the replication, partitioning, encryption and moving of data would be handled by some middle ware, this could also be used to keep an up-to-date abstract view of the system—this would not be location centric, but data centric, consequently not showing which hosts store which data, but which data is located at which hosts, in which status. The middle ware could also be used to keep a log of events that caused the system to take a certain action.

Technical Issues.

The example instantiation for our paradigm already discusses some possible elements in a realization of the paradigm. To extend this example one could consider using hypervisor techniques such as those implemented in the Xen architecture [1]. Virtualization could thus be used to separate parts of applications and data even if they are distributed to the same machine, using operating system mechanisms for ensuring protection.

6. RELATED WORK

In [24] an early example of using mobility specifically to counter an attack is described. Here a light-weight process migration is proposed as an end-to-end solution to Denial-of-Service attacks. The process under attack is moved within a specified overlay network.

Other directly related work has been covered in the sections above. Additionally, one can look at research in the area of modeling and analysis of systems, such as [5, 10, 8, 18, 25]. It should be noted that, in principle, each of these techniques could be used to replace the second component of the suggested realization of our paradigm.

Finally, an important related area is that of *self-healing* systems in which autonomy and adaptability is essential. See [13] for a critical overview of the state-of-the-art in self-healing systems with a special focus on security.

7. CONCLUSION AND FUTURE WORK

At the very core of businesses and society, information systems pervade our daily life. They are essential at all levels of actions, from meaningless "surfing" to decisions that potentially can influence the well-being of companies, economies, or even whole populations. In these information systems an entity's data is accessible from almost everywhere, often without the need of physical presence in the entity's perimeter.

This risk that data might be accessible without proper legitimation, has led to a wide range of access control mechanisms, which are supposed to restrict access to data.

In this work we propose a new security paradigm that aims at using the network's flexibility to move data and applications away from potential attackers:

> Instead of simply protecting data and applications by firewalls, encryption, and access control, we need a paradigm-shift towards systems that protect data and applications from direct access by unauthorized users.

We also present a possible realization of the proposed paradigm, based on recent advances in language-based security and analysis, where data and applications are partitioned ahead-of-time and can be moved automatically based on activity both in the network as well as the real world.

We believe that this kind of protection is urgently needed as a kind of overlay on top of the always-on networks we are dealing with today, where it becomes increasingly difficult to determine who has access to a location where sensitive data is being stored, thereby posing a risk for that data.

The suggested paradigm combines well-established techniques and new approaches from many different areas, such as program partitioning [15, 11, 23], static analysis [21], erasure policies [6], data tethers [2], and other security mechanisms such as [12].

As argued above, the mechanisms suggested can easily be incorporated in more high-level policies such as the four-eyes principle,

and other policies can certainly be added, possibly requiring addition of resources for sensing the environment.

Acknowledgement

We would like to thank the anonymous reviewers for their invaluable feedback on earlier drafts of this article, and the participants of the New Security Paradigms Workshop 2009 in Oxford for lively discussions. The shepherds Michael Locasto and Sean Peisert deserve special thanks for helping us understanding what we wrote and clarifying what we meant.

8. REFERENCES

[1] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. In *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 164–177, New York, NY, USA, 2003. ACM.

[2] M. Beaumont-Gay, K. Eustice, V. Ramakrishna, and P. Reiher. Information protection via environmental data tethers. In *Proceedings of the New Securiy Paradigms Workshop (NSPW 2007)*, 2007.

[3] M. Bishop. The Insider Problem Revisited. In *Proc. of New Security Paradigms Workshop 2005*, Lake Arrowhead, CA, USA, Sept. 2005. ACM Press.

[4] R. C. Brackney and R. H. Anderson, editors. *Understanding the Insider Threat*. RAND Corporation, Santa Monica, CA, U.S.A., March 2005.

[5] R. Chinchani, A. Iyer, H. Q. Ngo, and S. Upadhyaya. Towards a theory of insider threat assessment. In *Proceedings of the 2005 International Conference on Dependable Systems and Networks*, pages 108–117. IEEE Computer Society, 2005.

[6] S. Chong and A. C. Myers. Language-Based Information Erasure. In *CSFW '05: Proceedings of the 18th IEEE Computer Security Foundations Workshop (CSFW'05)*, pages 241–254, Washington, DC, USA, 2005. IEEE Computer Society.

[7] G. Cole. Nailing jelly to a wall: is it possible? Available from `http://graeme.woaf.net/otherbits/jelly.html`, last visited April 9, 2008.

[8] M. Dacier and Y. Deswarte. Privilege graph: an extension to the typed access matrix model. In *Proceedings of the European Symposium On Research In Computer Security*, 1994.

[9] T. Dimitrakos, F. Martinelli, P. Y. A. Ryan, and S. A. Schneider, editors. *Formal Aspects in Security and Trust, Fourth International Workshop, FAST 2006, Hamilton, Ontario, Canada, August 26-27, 2006, Revised Selected Papers*, volume 4691 of *Lecture Notes in Computer Science*. Springer, 2007.

[10] J. Gorski and A. Wardzinski. Formalising fault trees. In F. Redmill and T. Anderson, editors, *Achievement and Assurance of Safety: Proceedings of the 3rd Safety-critical Systems Symposium*, pages 311–328, Brighton, 1995. Springer.

[11] R. R. Hansen and C. W. Probst. Secure dynamic program repartitioning. In *Proceedings of the 10th Nordic Workshop in Secure IT-Systems (NordSec 2005)*, 2005.

[12] K. Kursawe and S. Katzenbeisser. Computing under occupation. In *Proceedings of the New Securiy Paradigms Workshop (NSPW 2007)*, 2007.

[13] M. E. Locasto. Self-Healing: Science, engineering, and fiction. In *Proceedings of the New Securiy Paradigms Workshop (NSPW 2007)*, 2007.

[14] A. C. Myers. JFlow: practical mostly-static information flow control. In ACM, editor, *POPL '99. Proceedings of the 26th ACM SIGPLAN-SIGACT on Principles of programming languages, January 20–22, 1999, San Antonio, TX*, ACM SIGPLAN Notices, pages 228–241, New York, NY, USA, 1999. ACM Press.

[15] A. C. Myers and B. Liskov. Protecting privacy using the decentralized label model. *ACM Transactions on Software Engineering and Methodology*, 9(4):41000442, Oct. 2000.

[16] R. D. Nicola, G. Ferrari, and R. Pugliese. KLAIM: a Kernel Language for Agents Interaction and Mobility. *IEEE Transactions on Software Engineering*, 24(5):315–330, May 1998.

[17] F. Nielson, H. R. Nielson, and C. L. Hankin. *Principles of Program Analysis*. Springer-Verlag, 1999.

[18] R. Ortalo, Y. Deswarte, and M. Kaâniche. Experimenting with quantitative evaluation tools for monitoring operational security. *IEEE Transactions on Software Engineering*, 25(5):633–650, 1999.

[19] C. W. Probst and R. R. Hansen. An extensible analysable system model. *Information Security Technical Report*, 13(4):235–246, 2008.

[20] C. W. Probst and R. R. Hansen. Analysing Access Control Specifications. In *Accepted for the Fourth International IEEE Workshop on Systematic Approaches to Digital Forensic Engineering (SADFE 2009)*, 2009.

[21] C. W. Probst, R. R. Hansen, and F. Nielson. Where can an insider attack? In Dimitrakos et al. [9], pages 127–142.

[22] F. B. Schneider. Enforceable security policies. *ACM Transactions on Information and System Security*, 3(1):30–50, Feb. 2000.

[23] D. Søndergaard, C. W. Probst, C. D. Jensen, and R. R. Hansen. Program partitioning using dynamic trust models. In Dimitrakos et al. [9], pages 170–184.

[24] A. Stavrou, A. D. Keromytis, J. Nieh, V. Misra, and D. Rubenstein. MOVE: An end-to-end solution to network denial of service. In *Proceedings of the Internet Society (ISOC) Symposium on Network and Distributed Systems Security (SNDSS)*, pages 81–96, 2005.

[25] L. Swiler, C. Phillips, D. Ellis, and S. Chakerian. Computer-attack graph generation tool. June 12 2001.

[26] Wikipedia. Moscow-washington hotline. Available from `http://en.wikipedia.org/wiki/Moscow-Washington_hotline`, last visited April 9, 2008.

[27] S. Zdancewic, L. Zheng, N. Nystrom, and A. C. Myers. Untrusted hosts and confidentiality: Secure program partitioning. In G. Ganger, editor, *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP-01)*, volume 35, 5 of *ACM SIGOPS Operating Systems Review*, pages 1–14, New York, Oct. 21–24 2001. ACM Press.

[28] S. Zdancewic, L. Zheng, N. Nystrom, and A. C. Myers. Secure program partitioning. *ACM Transactions on Computer Systems*, 20(3):283–328, Aug. 2002.