Type-Based Distributed Access Control vs. Untyped Attackers

Tom Chothia[1] and Dominic Duggan[2]

[1] Laboratoire d'Informatique (LIX), École Polytechnique (CNRS), 91128 Palaiseau Cedex France, tomc@lix.polytechnique.fr
[2] Department of Computer Science, Stevens Institute of Technology, Hoboken, NJ 07030, USA, dduggan@cs.stevens-tech.edu

Abstract. This paper considers the effect of untyped attackers inside a distributed system where security is enforced by the type system. In previous work we introduced the *Key-Based Decentralised Label Model* for distributed access control. It combines a weak form of information flow control with cryptographic type casts to allow data to be sent over insecure channels. We present our model of untyped attackers in a simplified version of this calculus, which we call mini-KDLM. We use three sets of type rules. The first set is for honest principals. The second set is for attackers; these rules require that only communication channels can be used to communicate and express our correctness conditions. The third set of type rules are used to type processes that have become corrupted by the attackers. We show that the untyped attackers can leak their own data and disrupt the communication of any principals that place direct trust in an attacker, but no matter what the attackers try, they cannot obtain data that does not include an attacker in its access control policy.

1 Introduction

Type systems can provide a lightweight method to ensure security properties of a given piece of code. Once checked, the program can be run, with few restrictions, in the knowledge that the guarantees of the type system will still hold. These guarantees can also be extended to distributed processes communicating across an untrusted network, as long as each trusted process is well-typed. This paper addresses the question of what happens when a number of these "trusted" processes ignore the security types with the aim of acquiring sensitive data and disrupting other principals.

In previous work [CDV03], we introduce the *Key-Based Decentralised Label Model* for distributed access control. It combines a weak form of information flow control with typed cryptographic operations. The motivation is to have a type system that ensures access control while giving the application the responsibility to secure network communications, and to do this safely. Hence, removing the need to force the user into a "one size fits all" security solution, which would have to be implemented in the trusted computing base. We are

planning an implementation of this system as an extension of Java, which will be called Jeddak. Ultimately, we would like a correctness result for untyped attackers in our planned KDLM extension of Java. However, this would require us to include in our model a number of implementational details, which would obstruct the explanation of our method of dealing with untyped attackers. This would also be difficult to present in the space of one paper, so we leave this as further work. Instead we show how safety in the face of untyped attackers can be proved for a cut-down version of our system, which we refer to as mini-KDLM. This calculus is simple enough to illustrate the ideas of our system, while still capturing the salient features of KDLM.

We show that the untyped attackers can leak their own data and disrupt the communication of any other principals that place direct trust in an attacker. However, no matter what the attackers try, they cannot obtain data that does not include an attacker in its access control policy. We achieve this result by using a type system with three sets of type rules. The first set allows principals to be well-typed in the KDLM style. Attackers have their own type rules that allow them to ignore the access control types. That attackers have type rules at all is down to the need to maintain the separation between base types and channel types (pretending that an integer has a channel type won't make it into a communication channel). The final set of type rules allow for names that have been misplaced in honest principals; we refer to processes that have been interfered with in this way as corrupt. These rules allow for one name to take the place of another name, with a different type, as long as both names originally included at least one attacker in their access control policy.

Our rules for attackers and corrupt processes require data not to be misplaced, unless it includes an attacker in its access control policy. So, we show the correctness of our system by showing that well-typed systems always reduce to well-typed systems. To assist us, we first prove a lemma: in a well-typed system, we may substitute one type for another and the system will remain well-typed, as long as both types originally included an attacker in their access control policy. We prove this lemma by showing that we can use the type rules for corrupt processes to type any sub-processes affected by the type change.

The contributions of this work are:

- A model of untyped attackers and a correctness proof for systems under attack in mini-KDLM. This is also a foundation stone for a much more complicated model of untyped attackers in our planned implementation of KDLM.
- Showing how distributed, untyped attackers can be dealt with using a different set of type rules for honest principals, attackers and principals corrupted by the attackers.

$v \in$ Value	$::= w, x, y, z$ Variable	$\mid a, b, c, n$ Channel name	
	$\mid k^+, k^-$ Key names	$\mid \{v\}_k$	Encrypted value
$P \in$ Principals	$::= P, P_1, P_2 ...$		
$LT \in$ Labelled Types	$::= LT, T^L$		
$R \in$ Process	$::=$ stop		Stopped process
	$\mid$ receive $v?x;\ R$		Message receive
	$\mid\ !R$		replication
	$\mid$ send $v_1!v_2$		Message send
	$\mid$ new$(a : LT); R$		New channel
	$\mid$ newkey $(k^+ : Enc(\vec{P})^{L_1},$		
	$k^- : Dec(\vec{P})^{L_2}); R$		New Keys
	$\mid\ (R_1 \mid R_2)$		Parallel composition
	$\mid$ encrypt $\{v_1\}_{v_2}\ as\ x; R$		Encryption
	$\mid$ decrypt $v_1\ as\ \{x\}_{v_2}; R$		Decryption
$N \in$ Network	$::=$ empty		Empty network
	$\mid\ P[R]$		Principal
	$\mid$ new$(a : LT); N$		Channel binding
	$\mid\ (N_1 \mid N_2)$		Wire

Fig. 1. Syntax of mini-KDLM

In Section 2, we review the KDLM type system for distributed access control and introduce the simplified version, mini-KDLM. Next, in Section 3 we introduce the model of untyped attackers. In Section 4, we show how type rules can be used to characterize principals that have been corrupted by an attacker. Section 5 proves the correctness of our system by way of a subject reduction result. Section 6 discusses related work and finally, Section 7 concludes and briefly discusses further work.

2 mini-KDLM

The Decentralized Label Model (DLM) [ML97] is a model of information flow control that was introduced by Myers and Liskov. This model avoids one undesirable aspect of classical information flow control - the need for some centrally defined lattice of information levels - by implicitly defining a lattice based on access control.

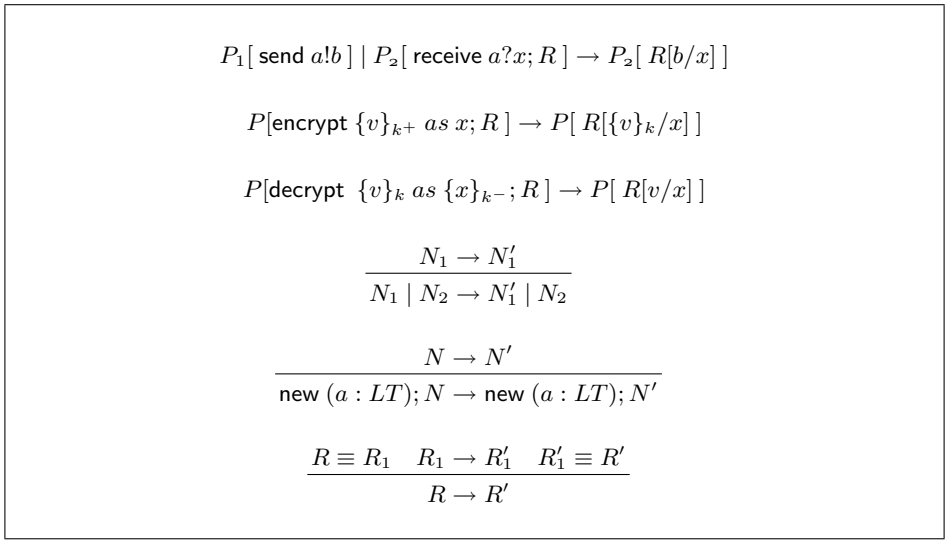

Fig. 2. The Semantics

More recently we combined ideas from DLM and cryptographic APIs [Dug03] to make the Key-based Decentralized Label Model (KDLM). This system provides distributed access control and forms the basis for mini-KDLM. The argument for our approach is the usual end-to-end argument in system design: it is ultimately unrealistic to expect there to be a single "one size fits all" solution to network security in the runtime. The application must be able to build its own network security stack for any approach to scale, so the type system prevents the application from violating the information flow while establishing network security.

The syntax of mini-KDLM is given in Figure 1. Most of this is similar to the spi-calculus [AG99]. The new term $P[R]$ is the process R running under the control of the principal P. It should be noted that this does not represent a location. It is possible to have two threads running for different principals on the same computer, just as it is possible for processes running for the same principal to run in two different places. We reduce and type each process running for a principal on its own, using the structural equivalence rule:

$$P[\ R_1 \mid R_2\ ] \equiv P[\ R_1\ ] \mid P[\ R_2\ ]$$

The calculus is monadic, meaning that channels only pass a single name at a time. We could extend the calculus to pass multiple channels at a time by repeating the type checks for each name passed, or by packaging up a number

$$!R \equiv R \mid !R \qquad \mathsf{stop} \mid R \equiv R$$
$$R_1 \mid R_2 \equiv R_2 \mid R_1 \qquad R_1 \mid (\, R_2 \mid R_3 \,) \equiv (\, R_1 \mid R_2 \,) \mid R_3$$
$$\mathsf{new}(a : LT); R \equiv R \quad a \notin fn(R) \qquad P[\, R_1 \mid R_2 \,] \equiv P[\, R_1 \,] \mid P[\, R_2 \,]$$
$$(\, \mathsf{new}(a : LT); R_1 \,) \mid R_2 \equiv \mathsf{new}(a : LT); (\, R_1 \mid R_2 \,) \quad a \notin fn(R_2)$$
$$\mathsf{new}(a_1 : LT_1); \mathsf{new}(a_2 : LT_2); R \equiv \mathsf{new}(a_2 : LT_2); \mathsf{new}(a_1 : LT_1); R$$

Plus the equivalent rules for Networks and newkey.

Fig. 3. Equivalence Rules

$T \in$ Types ::=	$\mathsf{Chan}(LT)$	Channel Type
|	$\langle\rangle$	Null Type
|	$Enc(\vec{P}) \mid Dec(\vec{P})$	Key Type
$L \in$ Label ::=	$\vec{P} \mid Public$	Access Control Policy
$LT \in$ Labelled type ::=	T^L	Protected Data

Fig. 4. Syntax of Sensitivity Types

of names into a single object and placing a policy on the object that is at least as restrictive as each of the names it contains.

The semantics of this calculus is given in Figure 2. This too, is similar to the spi-calculus, in particular, encrypting a name a with a key k results in the term $\{a\}_k$. This term cannot be identified as a, and cannot be used to communicate. The decrypt operation pattern matches the key name, and will decrypt the data if the correct key is provided, otherwise it will halt. The new construct generates a new and unique name. The structural equivalence rules allow the scope of a new name to be expanded as long as it does not capture any other names, using the rule:

$$(\mathsf{new}(a : LT); R) \mid R' \equiv \mathsf{new}(a : LT); (R \mid R') \quad \text{if } a \notin fn(Q)$$

where $fn(R')$ are the names in R' that do not appear under a binder. The communication rule cannot be applied across a new name construct hence new and "old" names represented by the same symbol cannot communicate. The other structural equivalence rules are given in Figure 3.

The access controls are enforced using the type system given in Figure 4 (syntax), Figure 5 (type rules) and Figure 6 (well-formed types). We do not enumerate the base types here, but they could include types such as int for integers, and $string$ for strings. The channel type $\mathsf{Chan}(LT)$ is the type of the communication channel that carries a value of type LT. A protected type adds

$$\frac{\Gamma \vdash N_1 \quad \Gamma \vdash N_2}{\Gamma \vdash (N_1 \mid N_2)} \qquad \frac{\Gamma \cup \{(a : LT)\} \vdash N}{\Gamma \vdash \mathsf{new}(a : LT)N}$$

$$\frac{\Gamma \vdash P[R_1] \quad \Gamma \vdash P[R_2]}{\Gamma \vdash P[(R_1 \mid R_2)]} \qquad \frac{\Gamma \cup \{(a : T^L)\} \vdash P[R] \quad P \in L \quad \vdash T^L}{\Gamma \vdash P[\mathsf{new}(a : T^L); R]}$$

$$\frac{\Gamma \vdash v : \mathsf{Chan}(T^L)^{L_0} \quad P \in L_0 \quad \Gamma \cup \{(x : T^L)\} \vdash P[R]}{\Gamma \vdash P[\mathsf{receive}\ v?x;\ R]}$$

$$\frac{\Gamma \vdash v_0 : \mathsf{Chan}(T^L)^{L_0} \quad \Gamma \vdash v : T^L \quad P \in L_0}{\Gamma \vdash P[\mathsf{send}\ v_0!v]}$$

$$\frac{\begin{array}{c}\vdash Enc(\vec{P})^{L_1} \quad \vdash Dec(\vec{P})^{L_2} \quad P \in L_1 \cap L_2 \\ \Gamma \cup \{k^+ : Enc(\vec{P})^{L_1}, k^- : Dec(\vec{P})^{L_2}\} \vdash P[R]\end{array}}{\Gamma \vdash P[\mathsf{newkey}\ (k^+ : Enc(\vec{P})^{L_1}, k^- : Dec(\vec{P})^{L_2}); R]}$$

$$\frac{\Gamma \vdash v_0 : T^L \qquad \Gamma \vdash v : Enc(L)^{L_k} \qquad \Gamma \cup \{x : T^{Public}\} \vdash P[R] \qquad P \in L_k}{\Gamma \vdash P[\mathsf{encrypt}\ \{v_0\}_v\ as\ x; R]}$$

$$\frac{\Gamma \vdash v_0 : T^{Public} \qquad \Gamma \vdash v : Dec(L_p)^{L_k} \qquad \Gamma \cup \{x : T^{L_p}\} \vdash P[R] \qquad P \in L_k}{\Gamma \vdash P[\mathsf{decrypt}\ v_0\ as\ \{x\}_v; R]}$$

Fig. 5. Types for Networks

a policy label to a channel or base type: T^L, for instance, $int^{\{Alice,Bob\}}$ is an integer that can only be used by the principals Alice and Bob. The aim of this type system is to ensure that names only ever reach principals that are mentioned in their policy, i.e., given a network with the term $P[R]$ all names used by the process R must name the principal P in their policy. In this section we direct the reader's attention to the basic calculus; we discuss encryption in the next section.

We restrict the types given to names in Figure 6 and we restrict how a process can use those names in Figure 5. Our type judgement on networks takes the form $\Gamma \vdash P[R]$ where Γ is a set of type bindings. The judgement on names $\Gamma \vdash a : LT$ means that a has type LT in Γ and LT is a well-formed type.

There are two fundamental restrictions imposed by the type system, the first is on channel types and the second is on the send action. Channels are required to have a policy that is more restrictive than the policy of the data they carry. This is enforced by the following rule from Figure 6.

$$\frac{v_0 : T^L \in \Gamma \qquad v : Enc(L)^{L_k} \in \Gamma \qquad \vdash T^L \qquad \vdash Enc(L)^{L_k}}{\Gamma \vdash \{v_0\}_v : T^{Public}}$$

$$\frac{\Gamma \vdash k^- : Dec(\vec{P})^{L'} \qquad k^+ : Enc(\vec{P})^L \in \Gamma \qquad \vdash Enc(\vec{P})^L}{\Gamma \vdash k^+ : Enc(\vec{P})^L}$$

$$\frac{\Gamma \vdash k^+ : Enc(\vec{P})^{L'} \qquad k^- : Dec(\vec{P})^L \in \Gamma \qquad \vdash Dec(\vec{P})^L}{\Gamma \vdash k^- : Dec(\vec{P})^L}$$

$$\frac{v : T^L \in \Gamma \quad \vdash T^L}{\Gamma \vdash v : T^L} \qquad \frac{\vdash T^L \quad L_0 \subseteq L}{\vdash Chan(T^L)^{L_0}} \qquad \frac{L \subseteq \vec{P}}{\vdash Dec(\vec{P})^L} \qquad \frac{L \subseteq \vec{P}}{\vdash Enc(\vec{P})^L}$$

Fig. 6. Well-Formed Types and Names

$$\frac{\vdash T^L \quad L_0 \subseteq L}{\vdash Chan(T^L)^{L_0}}$$

Here L_0 is the set of principals that can access a channel of this type and L is the set of principals that should be able to access the data sent across this channel. As a principal must possess a channel in order to be able to receive on it, the restriction $L_0 \subseteq L$ means that any restricted data can be sent over a correctly typed channel in the knowledge that any principal that can receive the data should be allowed to do so. The condition on the data type, $\vdash T^L$ ensures that this type is well-formed.

The type check on the send action, from Figure 5, ensures that only data of the correct type is sent over a channel, i.e., the type of v matches the type that should be carried by v_0.

$$\frac{\Gamma \vdash v_0 : \mathsf{Chan}(T^L)^{L_0} \quad \Gamma \vdash v : T^L \quad P \in L_0}{\Gamma \vdash P[\mathsf{send}\ v_0!v]}$$

As with the other type rules, this rule also checks that the types are well-formed and that all of the names can be used by the current principal. In the case of the send rule we know that, as v_0 has a well-formed type, $L_0 \subseteq L$ and hence the condition $P \in L_0$ implies that P is in the access control types for both v and v_0.

2.1 Encryption and Types

The type system described so far is very restrictive. More over, it may not always be possible to have a secure channel between any two principals. To make the

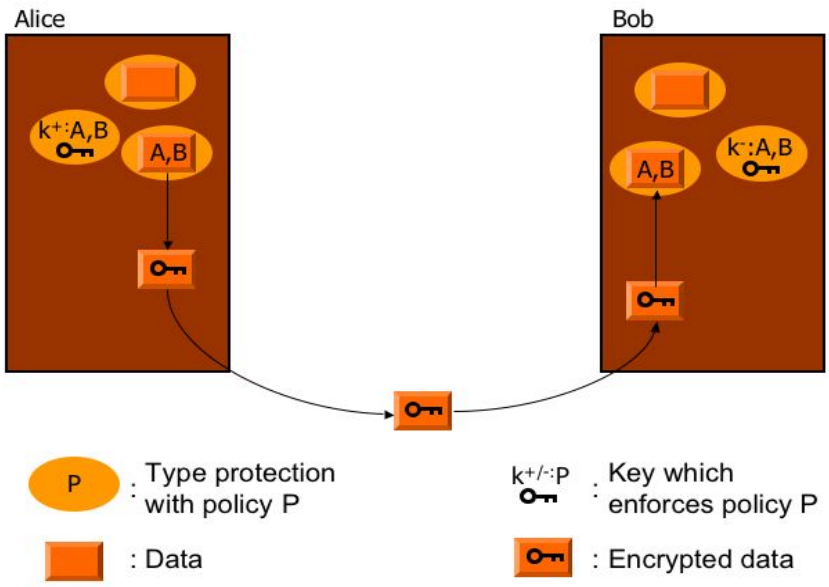

Fig. 7. Sending Data Through an Untrusted Area

type system more flexible we use encryption as a form of type downcasting to allow us to send sensitive data over an insecure channel, in a way that is both secure and type safe.

We associate access control lists with cryptographic keys. When a piece of data is sent over an insecure channel it is encrypted with a key that represents the list of principals that can access that data. Once encrypted, we remove the access control restriction from the data, indicated by the policy $Public$, meaning that the data is not unrestricted. We consider all principals to be in $Public$, so the test $P \in Public$ is always true. When encrypted data is received, the access restrictions for the decryption key are used as the access control type. Keys have the type $Enc(\vec{P})^L$ or $Dec(\vec{P})^L$ to represent an encryption key or a decryption key that enforces the policy $\vec{P}$ on data. These key types are in turn protected by a policy, in this case L. This is illustrated in Figure 7. In this picture, Alice wishes to send Bob some data, which is restricted to just the two of them. Lacking a secure channel she encrypts the data with a key that enforces the same policy as the one on the data, a type check ensures that these policies match. The encrypted data does not have any type restrictions and so can be sent to Bob over a public channel. Upon receiving it, Bob decrypts it and replaces the access restrictions. Hence, as long as the key is restricted to just Alice and

Bob, the data has passed from one principal to another in a safe way and has arrived with the same type as it started with.

The type rule for encryption is given in 5. This rule ensures that the right key is used to encrypt controlled data.

$$\frac{\Gamma \vdash v_0 : T^L \quad \Gamma \vdash v : Enc(L)^{L_k} \quad \Gamma \cup \{x : T^{Public}\} \vdash P[R] \quad P \in L_k}{\Gamma \vdash P[\mathsf{encrypt}\ \{v_0\}_v\ as\ x; R]}$$

We note that the policy on the data being encrypted (L) must match the policy that is enforced by the key. Once the name is encrypted the access control restrictions are removed, so it can then be sent over an insecure channel in a type safe way, this is indicated by the $Public$ label on the encrypted data. The condition that requires the current principal to be included in the policy of the key ($P \in L_k$) and the well-formedness condition on the key type imply that well-typed principals will only try to encrypt data that they are allowed to use i.e., that $P \in L$.

The matching decryption rule takes a name, without any access restrictions, and tries to decrypt it. If the incorrect key is used the process halts. If the correct key is provided we decode the data and give it the access control policy that is enforced by the key. As the decryption and encryption part of a key must enforce the same policy, we know that the decrypted name has the same type as it had before it was encrypted.

Keys are restricted to a subset of the principals in the policy they enforce, as seen in Figure 6. The well-formedness condition on the key types also ensures that encryption and decryption types for the same key enforce the same policy and then any encrypted terms in the initial network must also be well-typed, i.e., it must be possible to generate them from well-typed encryptions.

As an example, consider a system with two principals, a PDA and a $Base$ computer. If the PDA uses data packets of type $data$, then a packet that was restricted to just the PDA and the owners base computer would have the type $data^{\{PDA,Base\}}$, whereas public data would have the type $data^{\{Public\}}$.

Imagine that the PDA has a cable that connects it to the base computer and a wireless connection. The socket for the cable connection on the PDA will securely connect the PDA and base computer and so it would have a type indicating that it is safe for restricted data, $\mathsf{Cable\_Socket} : \mathsf{Chan}(data^{\{PDA,Base\}})^{\{PDA\}}$. The PDA label on the socket indicates that the socket connection cannot be sent to another location. The wireless connection, on the other hand could easily be intercepted. While it would be possible to use a secure transport layer to protect the data sent over this connection, this might be too great a burden for a the limited CPU and battery power of the PDA, or we might just want to keep the PDA

$$\frac{\Gamma \vdash_A N_1 \quad \Gamma \vdash_A N_2}{\Gamma \vdash_A (N_1 \mid N_2)} \qquad \frac{\Gamma \cup \{(a : LT)\} \vdash_A N}{\Gamma \vdash_A \mathsf{new}(a : LT)N}$$

$$\frac{\Gamma \vdash_A P[R_1] \quad \Gamma \vdash_A P[R_2]}{\Gamma \vdash_A P[(R_1 \mid R_2)]} \qquad \frac{\Gamma \cup \{(a : T^L)\} \vdash_A P[R] \quad P \in L \quad \vdash_A T^L}{\Gamma \vdash_A P[\mathsf{new}(a : T^L); R]}$$

$$\frac{\Gamma \vdash v : \mathsf{Chan}(T^L)^{L_0} \qquad P \in A \qquad L_0 \cap A \neq \{\} \qquad \Gamma \cup \{x : T^L\} \vdash_A P[R]}{\Gamma \vdash_A P[\mathsf{receive}\ v?x;\ R]}$$

$$\frac{\Gamma \vdash v_0 : \mathsf{Chan}(T_1^{L_1})^{L_0} \qquad \Gamma \vdash v : T_2^{L_2} \qquad \lfloor T_1 \rfloor = \lfloor T_2 \rfloor \qquad P \in A \qquad L_0 \cap A \neq \{\} \neq L_2 \cap A}{\Gamma \vdash_A P[\mathsf{send}\ v_0!v]}$$

$$\frac{\Gamma \vdash v_0 : T^L \qquad \Gamma \vdash v : Enc(L_p)^{L_v} \qquad \Gamma \cup \{x : T^{Public}\} \vdash_A P[R] \qquad L_p \cap A \neq \{\} \qquad L \cap A \neq \{\} \qquad P \in A}{\Gamma \vdash_A P[\mathsf{encrypt}\ \{v_0\}_v\ as\ x; R]}$$

$$\frac{\Gamma \vdash v_0 : T^{Public} \qquad \Gamma \vdash v : Dec(L_p)^{L_k} \qquad \Gamma \cup \{x : T_p^L\} \vdash_A P[R] \qquad L_k \cap A \neq \{\} \qquad P \in A}{\Gamma \vdash_A P[\mathsf{decrypt}\ v_0\ as\ \{x\}_v; R]}$$

Fig. 8. Types for Attackers

software as simple as possible. So, we give the socket a type indicating that it is not safe for private data, $\mathsf{Wireless\_Socket} : \mathsf{Chan}(data^{\{Public\}})^{\{PDA\}}$.

When the PDA software is compiled we automatically detect if the program tries to send any controlled data over the wireless connection without encrypting it first. So, once checked the lightweight PDA program can run without any restrictions and without a cumbersome security transport layer.

3 Untyped Attackers

The aim of this paper is the correct integration of untyped, trusted attackers into our model. If an attacker is mentioned in the access control policy of a name it can acquire that name and send it on anywhere it sees fit. So, an untyped attacker can always leak some restricted data, but we can show that this abuse of trust is not transitive. If Alice restricts her data to just herself and to Bob, and excludes Eve, who does not respect the access control types, then the data should be safe from Eve, even if Bob trusts her with other data and channels.

We include attackers into our type system by adding two new sets of type rules and a new form of type judgement. We write $\Gamma \vdash_A N$ to mean that the network N is well-typed given the fact that A is a set of principals that can perform attacks by ignoring access control types. Processes can now be typed with the original "honest" type rules, or by a set of type rules for attackers, or by another set of type rules for processes that have been corrupted by misinformation from an attacker.

The type rules that the attackers must conform to are given in Figure 8. It may seem odd to have type rules for untyped attackers however, these rules place no restrictions on access control types. So, more accurately these are un-access-control-typed attackers. The type rules do force attackers to respect the basic nature of the names. For instance, as communication channels must be supported by some kind of infrastructure, the attacker cannot turn an integer into a communication channel just by changing its type. We characterise the types the attacker can interchange by defining an erasure relation.

$$\begin{array}{rclcrcl} \lfloor \mathsf{Chan}(LT)^L \rfloor & = & \mathsf{Chan}(\lfloor LT \rfloor)^{Public} & \qquad & \lfloor \langle\rangle^L \rfloor & = & \langle\rangle^{Public} \\ \lfloor Enc(\vec{P})^L \rfloor & = & Enc(\vec{P})^{Public} & & \lfloor Dec(\vec{P})^L \rfloor & = & Dec(\vec{P})^{Public} \end{array}$$

As long as the attackers only substitute names with the same erasure type, they can do what they like. In particular, we point out that the type of a name being sent over a channel does not have to match the type that should be carried by that channel, and that the policy enforced by the encryption key used to encode a name does not have to match the policy on the name. This means that an encrypted name could be encrypted with what could be regarded as the wrong key. This is catered for by the following additional type rule for corrupted encrypted terms.

$$\frac{v_0 : T^L \in \Gamma \qquad v : Enc(L_p)^{L_k} \in \Gamma \qquad \vdash_A T^L \qquad \vdash_A Enc(L_p)^{L_k} \qquad L_p \cap A \neq \{\} \neq L \cap A}{\Gamma \vdash_A \{v_0\}_v : T^{Public}}$$

The attackers create "correct" types. This is because we do not consider attackers obtaining values created by other attackers as a security leak and if these names are passed to a genuine process then the real type of the name will not matter. The type rules also check that at least one attacker is named in each rule used. This is a correctness criterion that, in effect, states that the attackers have not been able to acquire any names that did not explicitly give access to an attacker. We show later that this correctness criterion is preserved by reduction.

$$\frac{\Gamma \vdash v : \mathsf{Chan}(T^L)^{L_0} \qquad \Gamma \cup \{(x : T^L)\} \vdash_A P[R] \qquad A \cap L_0 \neq \{\}}{\Gamma \vdash_A P[\mathsf{receive}\ v?x;\ R]}$$

$$\frac{\begin{array}{c}\Gamma \vdash v_0 : \mathsf{Chan}(T_1^{L_1})^{L_0} \qquad \Gamma \vdash_A v : T_2^{L_2} \qquad \lfloor T_1 \rfloor = \lfloor T_2 \rfloor \\ L_2 \cap A \neq \{\} \neq L_1 \cap A \qquad if\ L_0 \cap A = \{\}\ then\ P \in L_0\end{array}}{\Gamma \vdash_A P[\mathsf{send}\ v_0!v]}$$

$$\frac{\begin{array}{c}\Gamma \vdash v_0 : T^L \qquad \Gamma \vdash v : Enc(L_p)^{L_k} \qquad \Gamma \cup \{x : T^{Public}\} \vdash_A P[R] \\ A \cap L \neq \{\} \neq L_p \cap A \qquad if\ A \cap L_k = \{\}\ then\ P \in L_k\end{array}}{\Gamma \vdash_A P[\mathsf{encrypt}\ \{v_0\}_v\ as\ x; R]}$$

$$\frac{\Gamma \vdash v_0 : T^{Public} \quad \Gamma \vdash v : Dec(L_p)^{L_k} \quad \Gamma \cup \{x : T^{L_p}\} \vdash_A P[R] \quad A \cap L_k \neq \{\}}{\Gamma \vdash_A P[\mathsf{decrypt}\ v_0\ as\ \{x\}_v; R]}$$

Fig. 9. Types for the Corrupt

4 Corrupted Principals

While the attackers cannot acquire sensitive data, they can cause their data to be misplaced. Hence, if you trust an untyped attacker you run the risk of becoming corrupted. We formalise this with the rules in Figure 9.

We note that rather than having three separate sets of type rules, it would have been possible to have a single, complicated type rule for each piece of syntax. These rules would coalesce the conditions from each of the three type rules. For the sake of the reader's comfort, and our sanity, we decided to keep the rules simple.

The send rule may be corrupt in three ways: an attacker might have messed around with the communication channel that is being used to send the data, or the data that is being sent, or both. As the channel must have a well-formed type it is possible for an attacker to have access to the data being sent over the channel but not have access to the channel itself. This means that, if the send action is corrupt in any way, it must include an attacker in the payload type. We use an "if" statement to see if the communication channel may also be corrupt; if it cannot be, it must be in the right place, i.e., have the current principal in its policy. If an attacker has corrupted the data so the type of the channel's payload does not match the type of the name being sent then both must contain the name of an attacker.

In a similar way, if the attacker is named in any part of an encryption action it, must be named in the policy enforced by the key and in the policy of the name being encoded. If an attacker is not named in the access control policy for the key then the attacker cannot interfere with the key and so the current

principal must be mentioned. Of course, as the process has been corrupted, the policy enforced by the key does not have to match the policy on the data being encrypted.

5 Correctness

Our type system does not allow the attackers to possess data they are not supposed to have. For this reason, a well-typed system is one in which a leak has not yet occurred, as shown by the following lemma.

Lemma 1. *A well-typed network is correct, only names that explicitly allow access to an attacker appear outside their designated area:*

If $\Gamma \vdash_A P[R]$ *and* $R = C[\mathsf{send}\ v_1!v_2]$ *or* $R = C[\mathsf{receive}\ v_1?x; R']$ *or* $R = C[\mathsf{encrypt}\ \{v_1\}_{v_2}\ as\ x; R']$ *or* $R = C[\mathsf{decrypt}\ v_1\ as\ \{x\}_{v_2}; R']$ *and* $\Gamma' \vdash v_1 : T_1^{L_1}, v_2 : T_2{}^{L_2}$*, where* Γ' *is* Γ *extended with the types defined by* $C[\_]$ *then* $P \in L_1$ *or* $A \cap L_1 \neq \{\}$ *and* $P \in L_2$ *or* $A \cap L_2 \neq \{\}$*.*

Proof. By induction on the syntax of R. We inspect the type rules for each piece of syntax, observe that the conditions are fulfilled and apply the induction hypothesis to type the remaining process.

The result of our correctness result takes the form of a subject reduction proof; we show that well-typed systems reduce to well-typed systems. The type system allows any given piece of syntax to be typed as an honest process, or an attacker, or a corrupted process. This leads to multiple cases to check when assuming a process is well-typed. A more interesting issue is that one type might have been used to type a name in a given process and then a name of a different type could be substituted into its place. This will happen when an attacker sends a wrongly typed name over a channel to an honest process. In which case the honest process will become corrupt and we will have to type the resulting process with the type rule for corrupt processes.

We use the following lemma to show that the substitution of one type for another is allowed by the type rules for corrupt processes, as long there is an attacker in the policy of both names.

Lemma 2. *If* $\Gamma \cup \{x : T_1^{L_1}\} \vdash_A P[R]$ *then for all* $\vdash T_2^{L_2}$ *such that* $\lfloor T_1 \rfloor = \lfloor T_2 \rfloor$ *and* $A \cap L_1 \neq \{\}$ *and* $A \cap L_2 \neq \{\}$ *we have that* $\Gamma \cup \{x : T_2^{L_2}\} \vdash_A P[R]$

Proof. By induction on the syntax of R. We give the receive case as an example.

- $R \equiv \mathsf{receive}\ x?y;\ R'$
  By the assumption that $P[R]$ is well-typed, with the original type for x, we know that there exists T_3 and L_3 such that $T_1 = \mathsf{Chan}(T_3^{L_3})$ and that

$\Gamma \cup \{x : T_1^{L_1}, y : T_3^{L_3}\} \vdash_A P[R']$. As $\lfloor T_1 \rfloor = \lfloor T_2 \rfloor$ we know that $T_2 = \mathsf{Chan}(T_4^{L_4})$ for some T_4 such that $\lfloor T_3 \rfloor = \lfloor T_4 \rfloor$. By the well-formedness condition for channel types we know that $L_1 \subseteq L_3$ and $L_2 \subseteq L_4$ hence $A \cap L_3 \neq \{\}$ and $A \cap L_4 \neq \{\}$. So, we can apply the induction hypothesis to show that $\Gamma \cup \{x : T_2^{L_2}, y : T_4^{L_4}\} \vdash_A P[R']$. Noting that $A \cap L_2 \neq \{\}$ allows us to type $P[R]$ by the receive type rule for corrupt processes.

This lemma has proven the heart of our correctness result, however it remains to show that types only ever get mixed up when they include an attacker in their access control policy. We do this in our main theorem.

Theorem 1 (Subject Reduction). *If $\Gamma \vdash_A N$ and $N \rightarrow N'$ then $\Gamma \vdash_A N'$.*

Proof. By induction on the reduction $N \rightarrow N'$, for each reduction rule we consider each possible typing rule that could have been applied to type N. We then show that we can type N', using Lemma 2 whenever the process becomes corrupted.

And finally, we restate this as correctness:

Corollary 1. *Given a well-typed honest network $\Gamma \vdash N$, for any set of attackers A and any network N_A such that $\Gamma \vdash_A N_A$ the network $N \mid N_A$ cannot reduce to a state in which an attacker has a name that does not include an attacker in its access control policy.*

Proof. By Lemma 1 and Theorem 1.

6 Related Work

Mini-KDLM is designed to be simple enough to illustrate our correctness proof while still producing results that are relevant to full KDLM [CDV03]. So, naturaly mini-KDLM is a cut down version of full KDLM. Both have policy types on data and the key types that enforce a policy on the data they encrypt but full KDLM uses an abstraction of key names to represent policies, this allows for an accountable model of declassification. In mini-KDLM we restrict both encryption and decryption keys, however full KDLM splits the access control types into policies for security and authentication, meaning that encryption and signing keys can be made public. In other work we show how key names can be distributed and sketch how KDLM could be implemented as a type system for Java, which we refer to as Jeddak [CDV04].

Our work is partly inspired by the Distributed Label Model [ML97] this model was implemented as the language JFlow [Mye99]. The Jif/Split compiler

[ZZNM02,ZCZM03] allows a program to be annotated with trust information, the code is then split into a number of programs that can be run on different hosts. The partitioning preserves the original semantics of the program and ensures that hosts that are not trusted to access certain data cannot receive that data. Hennessy and Riely [HR99,RH99], have developed a type system that controls attackers in the Dpi-calculus. They allow attackers to ignore the type rules, as we do here, but they use dynamic type checks to ensure that honest principals do not become corrupted.

Much of the work on wide-area languages has focused on security, for example, providing abstractions of secure channels [AFG00,AFG99], controlling key distribution [CV99,CGG00], reasoning about security protocols [AG99,Aba97], etc. Abadi [Aba97] considers a type system for ensuring that secrecy is preserved in security protocols. Other work on security in programming languages has focused on ensuring and preventing unwanted security flows in programs [DD77,VS97,PC00]. Sabelfeld and Myers [SM02] provide an excellent overview of this work.

7 Conclusion and Further Work

We have provided a model of untyped attackers inside the Key-Based Decentralised Label Model and proved that these attackers can cause only limited damage. The model works by having three sets of type rules: the first for honest processes, the second for attackers and the third for processes that have been corrupted by an attacker. The type rules also contain checks that ensure no data, which is not designated as accessible to an attacker, leaks outside its area. We prove the correctness of our system by showing subject reduction.

It may be interesting to introduce a sub-typing relation for labelled types. For instance, allowing data to be sent over channels that should carry a more restrictive data type, and effectively up grading the data's security restrictions. Also, modelling corrupted types as sub-types of the original types may allow us to reduce the total number of type rules. However, this does not catch the different possible behaviours of honest participants and attackers and it may make the extension of the systems more cumbersome.

We hope that this work will be a base from which to prove that an implementation of Key-Based Decentralised access control in Java is also safe from untyped attackers. We also hope that this method can be applied to other type systems for distributed security.

References

[Aba97] Martin Abadi. Secrecy by typing in security protocols. In *Theoretical Aspects of Computer Science*, pages 611–638, 1997.

[AFG99] Martin Abadi, Cedric Fournet, and Georges Gonthier. Secure communications processing for distributed languages. In *IEEE Symposium on Security and Privacy*, 1999.

[AFG00] Martin Abadi, Cedric Fournet, and Georges Gonthier. Authentication primitives and their compilation. In *Proceedings of ACM Symposium on Principles of Programming Languages*, 2000.

[AG99] Martin Abadi and Andrew Gordon. A calculus for cryptographic protocols: The spi calculus. *Information and Computation*, 148(1):1–70, January 1999.

[CDV03] Tom Chothia, Dominic Duggan, and Jan Vitek. Type-based distributed access control. In *Computer Security Foundations Workshop*, Asilomar, California, June 2003. IEEE.

[CDV04] Tom Chothia, Dominic Duggan, and Jan Vitek. Principals, policies and keys in a secure distributed programming language. In *Foundations of Computer Security*, 2004.

[CGG00] Luca Cardelli, Giorgio Ghelli, and Andrew D. Gordon. Secrecy and group creation. In *Concurrency Theory (CONCUR)*. Springer-Verlag, 2000.

[CV99] Guiseppe Castagna and Jan Vitek. A calculus of secure mobile computations. In *Internet Programming Languages*, Lecture Notes in Computer Science. Springer-Verlag, 1999.

[DD77] D. E. Denning and P. J. Denning. Certification of programs for secure information flow. *Communications of the ACM*, 1977.

[Dug03] Dominic Duggan. Type-based cryptographic operations. *Journal of Computer Security*, 2003.

[HR99] Matthew Hennessy and James Riely. Type-safe execution of mobile agents in anonymous networks. In *Secure Internet Programming: Security Issues for Distributed and Mobile Objects*, Lecture Notes in Computer Science. Springer-Verlag, 1999.

[ML97] Andrew C. Myers and Barbara Liskov. A decentralized model for information flow control. In *Symposium on Operating Systems Principles*, 1997.

[Mye99] Andrew C. Myers. Jflow: Practical mostly-static information flow control. In *Proceedings of ACM Symposium on Principles of Programming Languages*, pages 228–241, 1999.

[PC00] Francois Pottier and Sylvain Conchon. Information flow inference for free. In *Proceedings of ACM International Conference on Functional Programming*, 2000.

[RH99] James Riely and Matthew Hennessy. Trust and partial typing in open systems of mobile agents. In *Proceedings of ACM Symposium on Principles of Programming Languages*, 1999.

[SM02] Andrei Sabelfeld and Andrew Myers. Language-based information-flow security. *IEEE Journal on Selected Areas in Communications*, 2002.

[VS97] D. Volpano and G. Smith. A type-based approach to program security. In *Proceedings of the International Joint Conference on Theory and Practice of Software Development*. Springer-Verlag, 1997.

[ZCZM03] Lantian Zheng, Stephen Chong, Steve Zdancewic, and Andrew C. Myers. Building secure distributed systems using replication and partitioning. In *IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 2003.

[ZZNM02] Steve Zdancewic, Lantian Zheng, Nathaniel Nystrom, and Andrew C. Myers. Secure program partitioning. *Transactions on Computer Systems*, 20(3):283–328, 2002.