Information Flow Secure CAmkES

Amit Goyal, Akshat Garg, Digvijaysingh Gour, R. K. Shyamasundar and G. Sivakumar
Department of Computer Science and Engineering,
Indian Institute of Technology Bombay, Mumbai, India

Keywords: Access Control, Confidentiality, Integrity, Information Flow Security, seL4, CAmkES, RWFM.

Abstract: Component Architecture for microkernel-based Embedded Systems (CAmkES) is a framework used to build embedded systems software on the top of seL4. seL4, a general purpose microkernel, uses the underlying Discretionary Access Control (DAC) capability model to ensure confidentiality and integrity of the systems built on it. These systems are not information flow secure as DAC model only considers direct read/write accesses and does not consider the indirect accesses. In indirect access, an unauthorized subject can get access to an object through another subject which has the direct access to that object. In this paper, we model and implement information flow secure CAmkES (IFS-CAmkES) which ensures complete mediation by RWFM monitor which is based upon Readers Writers Flow Model (RWFM), a Mandatory Access Control (MAC) model. IFS-CAmkES can be considered as CAmkES enriched with MAC based security. Prototypes of some real life examples have been implemented on IFS-CAmkES. We also compare the performance of CAmkES and IFS-CAmkES based systems.

1 INTRODUCTION

In today's digital world, security is one of the major aspects that needs to be addressed, even though, in the operating system (in use), there is no clear layer for this attribute (Lampson, 2011). Additionally, in the context of Internet of Things (IoT), consisting of interconnected embedded systems, security cannot be thought of as an add-on to a device, but rather integral to the reliable functioning of the device. Software security controls need to be introduced at the operating system level which takes off the onus from device designers and developers to configure systems to mitigate threats, and ensure their platforms are safe. Further, there has always been a trade off between security and functionality (Jaeger, 2008). Most of the secure systems are either hard or not usable in practice because of the limited functionality.

There have been several attempts towards proving security properties of the operating systems but most of the proofs are either incomplete, require manual intervention or work at the high level abstraction and not at the implementation level (Klein et al., 2014). seL4 is a general purpose microkernel which has been fully formally machine verified and proofs carry through the high level abstractions down to the C code implementation (Klein et al., 2009). seL4 has small Trusted Computing Base (TCB) which makes it embedded system friendly, and its verification feasible (Heiser et al., 2007). Its strong isolation enforcement between mutually distrusting components, running on its top, makes it a perfect candidate to be used as a hypervisor (to host virtual machines) (Klein et al., 2018).

Systems built on seL4 have been verified to provide confidentiality, integrity, authority confinement and availability (Klein et al., 2014) (Elkaduwe et al., 2008). These systems only prevent the unauthorized direct read/write accesses of a subject to an object (owing to the capability based, Discretionary Access Control (DAC) implementation of seL4) but do not prevent the indirect accesses which lead to the information leak. In indirect access, an authorized subject, having direct access to an object, helps an unauthorized subject to get access to that object. This might involve multiple intermediary subjects and objects which ultimately aid an unauthorized subject in getting the access.

Component Architecture for microkernel-based Embedded Systems (CAmkES) is a platform which aids in building embedded systems software on seL4. It abstracts the low level mechanisms of the microkernel, and allows to define components and connections (between the components for communication). CAmkES based systems use the underlying seL4 se-

Table 1: Information leak examples.

Case	C1	C2	C3	X
1	Client 1	Helper	Client 2	Confidential Data
2	Bidder 1	Auctioneer	Bidder 2	Bid
3	Voter 1	Voting Machine	Voter 2	Vote

curity mechanism (DAC) and are thus not information flow secure. Consider an example where a GPS tracking device of a car sends the source and destination details to a navigation server which in return provides the directions. The navigation server might leak the location details to an intruder. Similarly, information leak might happen in the cases listed in Table 1 where Component 1 (C1) sends information 'X' to Component 2 (C2) which then sends (leaks) it to unauthorized Component 3 (C3).

The question which arises is, can we synthesize information flow secure, CAmkES based systems by augmenting a Mandatory Access Control (MAC) model on the existing DAC model provided by CAmkES? In an attempt to answer this, we augment CAmkES with Readers Writers Flow Model (RWFM), a MAC model and coin the new framework "Information Flow Secure CAmkES" (IFS-CAmkES) as the systems built on it will be information flow secure. IFS-CAmkES ensures complete mediation by RWFM monitor on every read/write access. The main contributions of this paper are: *(i)* modeling and implementation of IFS-CAmkES, *(ii)* implementation of some real life examples (prototypes) on IFS-CAmkES to justify its application, *(iii)* performance comparison of CAmkES and IFS-CAmkES based systems. To the best of our knowledge, no previous work has integrated MAC based security in CAmkES to make information flow secure, CAmkES based systems.

The rest of the paper is organised as follows: Section 2 provides a review of the various attempts towards secure operating system and a brief description of CAmkES, relevant to this study. RWFM is discussed in Section 3. IFS-CAmkES model is presented in Section 4. Section 5 provides its implementation details, its application in real life examples, and performance comparison of IFS-CAmkES and CAmkES based systems. Finally, we conclude in Section 6 along with the future directions.

2 BACKGROUND

In this section, we provide a brief literature review on efforts to build secure operating system, and briefly describe features of CAmkES.

2.1 Attempts towards Secure Operating System

Verification of security properties has been one of the important objectives of the operating system verification. Early work started with UCLA Secure Unix which mainly focused on correctness, and Provably Secure Operating System (PSOS) which focused on formal kernel design but the proofs were not completed by both (Walker et al., 1980) (Feiertag and Neumann, 1979).

Linux is susceptible to trojan horse and information flow leaks as it is based on DAC. SELinux implemented as a Linux security module is one of the attempts to make Linux more secure (Loscocco and Smalley, 2001) (Smalley et al., 2001). Proofs for verifying information flow goals in SELinux, work only at higher level kernel abstractions and not at the code level (Guttman et al., 2005). Similarly, EROS kernel and the MASK project proofs do not proceed to the implementation level (Farber and Smith, 1996) (Martin et al., 2000).

Flume, a capability based operating system, considers the entire Linux kernel in its TCB. Further, non interference proof is done manually (Krohn and Tromer, 2009). INTEGRITY-178B provides the proof for isolation and information flow but the proof is manually connected to the source code (Richards, 2010).

To enforce information flow control, HiStar implements simple semantics based on object labels and category ownership. However, there is no proof to show that the semantics correctly model the behaviour of its implementation (Zeldovich et al., 2011).

seL4 is a general purpose microkernel that has an automated and full proof of its functional correctness from the high level specification to the C code implementation (Klein et al., 2014). Its capability based access control correctly models the behaviour of its implementation. It can be configured to enforce static information flow security in the form of intransitive non interference. In intransitive non interference if the information is allowed to flow from A to B and B to C, then if the information flows from A to C it must flow via B.

Systems built on seL4 have been formally verified for providing the following security properties:

(i) **Availability:** An unauthorized application should not be able to deny service in terms of the resources (e.g. processor time and memory resources) that the kernel manages (Klein et al., 2014). *(ii)* **Authority confinement:** Without explicit authorization, the authority cannot be escalated or transferred to another entity (Sewell et al., 2011). *(iii)* **Integrity:** Without authorization, an application cannot write to or change resource state (Sewell et al., 2011). *(iv)* **Confidentiality:** No unauthorized read operations can be performed (Murray et al., 2013).

seL4 is a capability based system (DAC) and systems built on it provide security only from direct accesses and not from indirect accesses. Thus, integrity and confidentiality specifications consider only direct write and read operations respectively.

2.2 CAmkES

CAmkES is an architecture to develop embedded systems software on the top of seL4 (Kuz et al., 2007). It abstracts over the low-level kernel mechanisms and allows us to define components, connections between the components and interfaces through which the components communicate over the connections. The components can be active or passive depending on whether it has a control thread or not. CAmkES provides 4 types of connections:

- **RPC Connection** is used for making remote procedure calls via from-interface instance (of the caller) to the to-interface instance (of the callee). An interface may contain multiple procedures. Components either provide (to-interface instance) or use (from-interface instance) the interface. We have used the terms interface and interface instance interchangeably.
- **RPC Call Connection** is the extension of RPC connection in which callee also replies back to the caller. Both RPC and RPC Call are implemented using seL4's endpoint objects.
- **Event Connection** is used for providing event notifications between the components. It is implemented using seL4 notification mechanism.
- **Dataport Connection** allows components to communicate over shared pages.

In CAmkES, the entire system is defined as an assembly composition. The CAmkES parser first generates the Abstract Syntax Tree (AST) for the CAmkES assembly composition. There are templates for interfaces and components which are then used by the parser to produce the glue code (implementation) which is their seL4 level code [1] (Klein et al., 2018).

Figure 1 is an example of CAmkES based system consisting of three components Client 1 (C1), Helper (H) and Client 2 (C2). C1 is connected to H via RPC connection (h1) and H is connected to C2 via RPC connection (h4). Interface instances have also been shown below the arrows. From the arrow direction, we can observe, h2 is the from-interface (use) instance and h3 is the to-interface (provide) instance. The assembly description for the system is given in Listing 1.

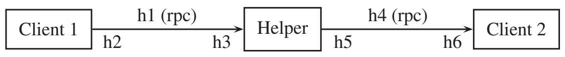

Figure 1: CAmkES based system.

Listing 1: Assembly description.

```
assembly {
  composition {
    component Client_1 Client1;
    component Client_2 Client2;
    component Help Helper;
    connection seL4RPC h1(from Client1.h2, to Helper.
         h3);
    connection seL4RPC h4(from Helper.h5, to Client2.
         h6);
  }
}
```

Here, we can clearly see, information is only allowed to flow from C1 to H and from H to C2. But indirectly, it can flow from C1 to C2 also (which is not specified in the composition). We use RWFM (MAC) to prevent such indirect flows in CAmkES based systems and present IFS-CAmkES. It is important to note that we assume, the connections in the assembly composition specify the access control policy. IFS-CAmkES considers all other flows which are not mentioned in the assembly composition as forbidden (never allowed). Section 3 provides a brief description of RWFM.

3 RWFM

RWFM (Kumar and Shyamasundar, 2014) (Kumar and Shyamasundar, 2017) takes into account the Denning's information flow control model (Denning, 1976) and suggests a natural way of defining the security classes. It is a lattice based model which ensures both confidentiality and integrity, and can thus capture the behavior of popular information flow control models like Biba (Biba, 1977) and Bell-LaPadula

[1] More details on structure and functionality of CAmkES can be found at: https://github.com/seL4/camkes-tool/blob/master/docs/index.md

(Bell and LaPadula, 1973). RWFM is represented as an eight tuple $< S, O, SC, \rightarrow, \oplus, \otimes, \top, \bot >$ where:

- *S* are subjects and *O* are objects.
- *SC* (security classes/labels) are defined as S x 2^S x 2^S (owner x {set of readers} x {set of writers}). These labels form a lattice (on permissible flow ordering) and are assigned to all the subjects and objects by the labelling function λ. Object labels are static while subject labels are dynamic. Owner is required only while considering the downgrade operation and has nothing to do with other operations. We can get first, second and third components of subject/object label using the functions: Owner(), Readers() and Writers() respectively.
- $\rightarrow$ (permissible flow ordering) is defined as (-,$\supseteq$, $\subseteq$). Information always flows up in the lattice.
- $\oplus$ (join) is defined as (-, $\cap$, $\cup$) and $\otimes$ (meet) is defined as (-, $\cup$, $\cap$).
- $\top$ (maximum label) is defined as (-, ϕ, S) and $\bot$ (minimum label) is defined as (-, S, ϕ).

The default/initial label for a subject s is (s, S, {s}) and the objects labels are assigned based on the access rights. Rules (determine permissible flow of information) are defined for the basic operations as follows:

- Read Rule: When a subject (s) with label (s1, r1, w1) wants to read from an object (o) with label (s2, r2, w2) then: *(i)* s must belong to r2, *(ii)* the label of s should be updated to the join of the labels of s and o as s has gained information.

  Algorithm 1 defines the read rule. It takes subject (s) and object (o) as inputs, and returns whether read operation is allowed or not. Once the read operation is complete, the subject label is updated to **(s1, r1 $\cap$ r2, w1 $\cup$ w2)**.

Algorithm 1: Read rule.

```
Input: s, o
Output: Read is allowed or not
1 if $s \in r2$ then
2     return 1;
3 end
4 else
5     return 0;
6 end
```

- Write Rule: When a subject (s) with label (s1, r1, w1) wants to write on an object (o) with label (s2, r2, w2) then: *(i)* s must belong to w2, *(ii)* the label of s must be lower in the lattice than o as the information is flowing from s to o.

  Algorithm 2 defines the write rule. It takes subject (s) and object (o) as inputs, and returns whether write operation is allowed or not.

Algorithm 2: Write rule.

```
Input: s, o
Output: Write is allowed or not
1 if $s \in w2 \wedge r1 \supseteq r2 \wedge w1 \subseteq w2$ then
2     return 1;
3 end
4 else
5     return 0;
6 end
```

RWFM helps in preventing indirect read and write accesses in the system. The detailed model for IFS-CAmkES is presented in Section 4.

4 INFORMATION FLOW SECURE CAmkES MODEL

Out of the four types of connections in CAmkES based systems, we have secured the information flow on RPC and RPC Call connections. Dataport connection could not be secured because in CAmkES, dataport access rights are frozen in page table of the process (component), and then access is completely controlled by the hardware (MMU) during execution. Thus, it would not be feasible to trace the read and write operations at the software level and use RWFM. Moreover, dataport write access is implemented as both read and write accesses, which is inconsistent even with the direct access (due to the extra read access). Event connection is used for single bit communication in CAmkES, so we have not considered it.

To secure RPC and RPC Call connections using RWFM, we have not used shared variables to store the RWFM labels as this may lead to inconsistency in the labels, and the labels may be misused, resulting in information leaks. Also, we have not changed the existing system calls or added new system calls for RWFM checks (rules) as this might interfere with the underlying seL4 proofs.

We have added an another component called RWFM Monitor at the CAmkES level. It is connected with all other components of the system using RPC Call connections. To prevent the information leak from client 1 to client 2 in Figure 1, RWFM monitor is used as shown in Figure 2.

The initials labels for subjects (components) and objects (interfaces) are generated from the AST and are fed into the RWFM monitor which stores and manages these labels. On its interface instances, RWFM monitor provides three procedures: can_i_read(), can_i_write() and update_my_label(). can_i_read() is same as Algorithm 1 except that it

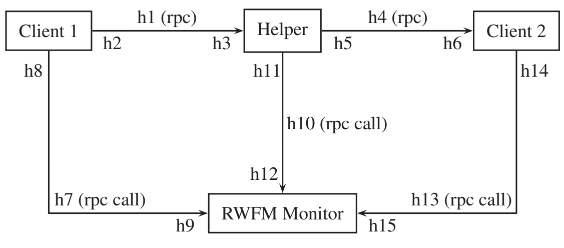

Figure 2: RWFM monitor.

only takes one input i.e. the object. The subject is not needed since the monitor can infer it from the interface used for the call. Similar argument holds for can_i_write and Algorithm 2. update_my_label updates the subject's label on successful read operation. RPC and RPC Call interface templates are modified to ensure RWFM monitor completely mediates all the read/write requests during RPC and RPC Calls between the components. The details are presented in Section 4.1-4.3.

Since CAmkES allows us to define components and connections over a single system we have assumed that all the components exist as virtual machines over a single system (with seL4 as hypervisor). In general, even if the components do not exist on a single system the communication between the sub-components of a component might lead to information leak and to prevent it, we can apply RWFM at the sub-component level. Further, information leak over the network can be prevented by applying RWFM monitor on the access control policy of the network.

4.1 Label Generation

Label generation is done using Algorithm 3 which takes AST as the input and generates the RWFM labels which are then stored in the RWFM monitor. Labels are generated for the components (subjects) and interfaces (objects). *C*, *I* and *N* represent the set of components, interfaces and connections respectively in the original system (without including RWFM monitor).

The algorithm assigns the default label to all the components. Initially, for each interface, the parent component is assigned as the owner, and readers and writers set are kept empty.

For RPC connection, the parent component of the from-interface acts as the writer and is thus added in the writers set of both the interfaces of the connection. The parent component of the to-interface acts as the reader and is thus added in the readers set of both the interfaces of the connection.

For RPC Call connection, the parent components of both the interfaces act as readers and writers (both)

Algorithm 3: Label generation.

```
Input: AST
Output: Generates the RWFM labels.
1  for each component $c_x \in C$ do
2      $\lambda(c_x)$ = ($c_x$, C, {$c_x$})
3  end
4  for each interface $i_y \in I$ do
5      $\lambda(i_y)$ = (parent component, {}, {})
6  end
7  for each connection $n_z \in N$ do
8      if $n_z$_type == rpc then
9          Readers($n_z$_to_interface) =
             Readers($n_z$_to_interface) $\cup$
             $n_z$_to_interface_parent_component
             Writers($n_z$_to_interface) =
             Writers($n_z$_to_interface) $\cup$
             $n_z$_from_interface_parent_component
             Readers($n_z$_from_interface) =
             Readers($n_z$_from_interface) $\cup$
             $n_z$_to_interface_parent_component
             Writers($n_z$_from_interface) =
             Writers($n_z$_from_interface) $\cup$
             $n_z$_from_interface_parent_component
10     end
11     if $n_z$_type == rpc call then
12         Readers($n_z$_to_interface) =
             Readers($n_z$_to_interface) $\cup$
             $n_z$_to_interface_parent_component $\cup$
             $n_z$_from_interface_parent_component
             Writers($n_z$_to_interface) =
             Writers($n_z$_to_interface) $\cup$
             $n_z$_to_interface_parent_component $\cup$
             $n_z$_from_interface_parent_component
             Readers($n_z$_from_interface) =
             Readers($n_z$_from_interface) $\cup$
             $n_z$_to_interface_parent_component $\cup$
             $n_z$_from_interface_parent_component
             Writers($n_z$_from_interface) =
             Writers($n_z$_from_interface) $\cup$
             $n_z$_to_interface_parent_component $\cup$
             $n_z$_from_interface_parent_component
13     end
14 end
```

and are thus added in both the readers and writers set of both the interfaces of the connection.

Note that no labels are assigned to RWFM monitor, its interfaces and any interface involved in communication with the RWFM monitor.

The labels generated for the system shown in Figure 1 are as follows:
λ(C1) = (C1,{C1,H,C2},{C1})
λ(H) = (H,{C1,H,C2},{H})
λ(C2) = (C2,{C1,H,C2},{C2})
λ(h2) = (C1,{H},{C1})
λ(h3) = (H,{H},{C1})
λ(h5) = (H,{C2},{H})
λ(h6) = (C2,{C2},{H})

If both the connections in Figure 1 are replaced by RPC Call connections then the labels of the interfaces are as follows:
λ(h2) = (C1,{C1,H},{C1,H})
λ(h3) = (H,{C1,H},{C1,H})
λ(h5) = (H,{C2,H},{C2,H})
λ(h6) = (C2,{C2,H},{C2,H})

4.2 RPC Template

RPC to-interface template has seL4_Recv() system call to read (receive) the data. It is modified to do RWFM read check as shown in Listing 2. If it fails, the data is not read. Otherwise, the component label is updated once the read operation is completed.

Listing 2: RPC to-interface template.

```
if (!can_i_read(interface);)
        return;
seL4_Recv();
update_my_label();
```

RPC from-interface template has seL4_Send() system call to send the data. It is modified to do RWFM write check as shown in Listing 3. If it fails, the data is not sent.

Listing 3: RPC from-interface template.

```
if (!can_i_write(interface);)
        return;
seL4_Send();
```

Note that, can_i_read(), can_i_write() and update_my_label() will be called as h11_can_i_read(), h11_can_i_write() and h11_update_my_label() respectively, by H in all its RPC to/from-interfaces implementations (seL4 code). C1 and C2 also use their own interfaces, h8 and h14 respectively as shown in Figure 2. The argument for these procedures will be the interface whose implementation is being done. As an example, for h5 interface (RPC from-interface) implementation, h11_can_i_write(h5) is used.

Now, when C1 makes an RPC to H, RWFM write check for C1 at h2 is performed which succeeds. RWFM read check for H at h3 also succeeds and the label of H is updated to: λ(H) = (H,{H},{C1,H}). Now, when H makes an RPC to C2, RWFM write check for H at h5 fails and thus information leak (indirect write from C1 to C2) is avoided.

4.3 RPC Call Template

RPC Call to-interface template has seL4_Recv() system call to receive the data and seL4_Reply() system call to send the data (reply back to the caller). RWFM read check is added before seL4_Recv() as shown in Listing 4. If it fails, the data is not received. Otherwise, the component label is updated once it receives. Further, RWFM write check is added before seL4_Reply().

Listing 4: RPC Call to-interface template.

```
if (!can_i_read(interface);)
        return;
seL4_Recv();
update_my_label();
if (!can_i_write(interface);)
        return;
seL4_Reply();
```

RPC Call from-interface template has seL4_Call() system call to send the data and receive the reply. It is modified to do RWFM write check as well as read check as in shown in Listing 5. If any of the check fails, the data is not sent and the reply is not received. Otherwise, the component label is updated once it receives the reply.

Listing 5: RPC Call from-interface template.

```
if (!can_i_write(interface);)
        return;
if (!can_i_read(interface);)
        return;
seL4_Call();
update_my_label();
```

It is important to note that the object (interface) labels do not change during execution and they contain the access control policy which is derived from the connections (RPC/RPC Call) specified in the assembly composition. Section 5 provides the implementation details.

5 IMPLEMENTATION DETAILS

We have implemented IFS-CAmkES by modifying CAmkES 3.5.0 (built on seL4 10.0.0). We used Intel Core i7, 8th generation machine with 16 GB of RAM and Ubuntu 16.04 LTS operating system. We used QEMU emulator version 2.5.0 to run CAmkES/IFS-CAmkES.

The example stated in Figure 1 when implemented on IFS-CAmkES, stops the indirect write by client 1 to client 2 via a helper (preserves integrity of client 2 data). Similarly, the example stated in Figure 3 when implemented on IFS-CAmkES, stops the indirect read by client 1 from client 2 via a helper (preserves confidentiality of client 2 data).

In auctioning example shown in Figure 4, an auctioneer accepts the bids from bidder 1, bidder 2 and

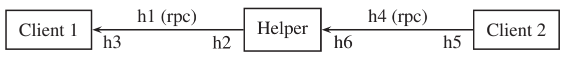

Figure 3: Client 1 indirectly reads from client 2.

bidder 3. The connections from bidders to auctioneer are used for making the bid while the connections from auctioneer to bidders are used for declaring the result. All the connections are RPC connections. Once the auctioneer starts getting the bids, he should not convey those bids to other bidders, and should not declare the result apriori. We cannot assure these conditions on CAmkES while on IFS-CAmkES we can. Once the auctioneer gets a bid, his label gets updated in such a way, that he cannot write to other bidders. For declaring the result, auctioneer can request the RWFM monitor to be turned off and only then the result can be declared. Also, we can implement RWFM at the language level (where each variable is given a RWFM label and every operation on the variable happens as per the information flow policy) and apply declassification on the *result* variable in auctioneer in order to declare the result. Similar scenario holds for the electronic voting system (replace auctioneer by voting machine, bidders by voters and bids by votes).

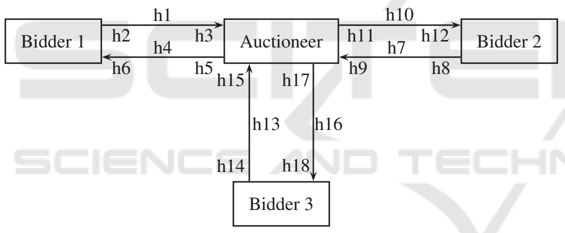

Figure 4: Auctioning.

In GPS tracking example shown in Figure 5, GPS tracking device of a car makes an RPC Call (using connection h1) to a navigation server with source and destination as arguments, and navigation server returns the directions. The navigation server may leak these location details to an intruder via RPC connection h4. When implemented on IFS-CAmkES such a leak can be avoided.

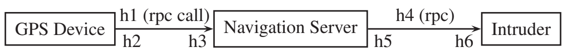

Figure 5: GPS tracking.

We made a simple system consisting of only two components and made a single RPC/RPC Call connection between them and measured the number of RPC/RPC Call communications that happened between the two components in 100 seconds (both on CAmkES and IFS-CAmkES). The experiment was performed multiple times to find the average. Table 2 shows the results.

Table 2: Performance comparison (in terms of number of RPC/RPC Call communications in 100 seconds).

	RPC	RPC Call
IFS-CAmkES	195872	153056
CAmkES	310615	403036

It is observed that the performance (with respect to time) of IFS-CAmkES based system as compared to that of CAmkES, reduces by approximately 37% in case of RPC and by 62% in case of RPC Call. This is due to the extra RPC Calls made by the components (to the RWFM monitor for performing the RWFM checks) involved in the communication. The reduction in performance in case of RPC Call communication is higher than that of RPC as the former makes more RPC Calls to the RWFM monitor. To improve the performance, we can implement RWFM at the seL4 level where RWFM checks can be performed inside the system calls. This would require us to redo the seL4 correctness proofs in order to ensure that the correctness still holds.

6 CONCLUSION AND FUTURE SCOPE

Capability based (DAC) systems built using CAmkES on seL4, cannot prevent the indirect accesses and thus cannot prevent information leak in the system. To capture the information leak, one has to implement the labelled MAC. To make CAmkES based systems information flow secure, we have proposed a model (inspired from RWFM) which generates initial labels based on components, RPC and RPC Call connections (assuming the connections specify the access control policy) specified in the CAmkES assembly composition. All read/write accesses in the system are completely mediated by the RWFM monitor which also updates the label of the subject when a read operation happens. The proposed model has been successfully implemented and integrated in CAmkES. We term this modified framework as Information Flow Secure CAmkES (IFS-CAmkES) as the systems built on it are information flow secure. From application point of view, we have implemented, real life examples (prototypes) like auctioning, electronic voting, GPS tracking, etc., on IFS-CAmkES. Although, the performance (with respect to time) of IFS-CAmkES based systems is less than that of CAmkES but there is always a trade off between security and performance.

To overcome the performance penalty and to provide information flow security at the microkernel level, we can implement RWFM at the level of seL4. To introduce information flow security restrictions at a fine grained level, language level RWFM implementation can also be considered.

REFERENCES

Bell, D. E. and LaPadula, L. J. (1973). Secure computer systems: Mathematical foundations. Technical report, MITRE CORP BEDFORD MA.

Biba, K. J. (1977). Integrity considerations for secure computer systems. Technical report, MITRE CORP BEDFORD MA.

Denning, D. E. (1976). A lattice model of secure information flow. *Communications of the ACM*, 19(5):236–243.

Elkaduwe, D., Klein, G., and Elphinstone, K. (2008). Verified protection model of the sel4 microkernel. In *Working Conference on Verified Software: Theories, Tools, and Experiments*, pages 99–114. Springer.

Farber, D. J. and Smith, J. M. (1996). State caching in the eros kernel–implementing efficient orthogonal persistence in a pure capability system. In *Proceedings of 7th International Workshop on Persistent Object Systems*. Citeseer.

Feiertag, R. J. and Neumann, P. G. (1979). The foundations of a provably secure operating system (psos). In *1979 International Workshop on Managing Requirements Knowledge (MARK)*, pages 329–334. IEEE.

Guttman, J. D., Herzog, A. L., Ramsdell, J. D., and Skorupka, C. W. (2005). Verifying information flow goals in security-enhanced linux. *Journal of Computer Security*, 13(1):115–134.

Heiser, G., Elphinstone, K., Kuz, I., Klein, G., and Petters, S. M. (2007). Towards trustworthy computing systems: Taking microkernels to the next level. *ACM SIGOPS Operating Systems Review*, 41(4):3–11.

Jaeger, T. (2008). Operating system security. *Synthesis Lectures on Information Security, Privacy and Trust*, 1(1):1–218.

Klein, G., Andronick, J., Elphinstone, K., Murray, T., Sewell, T., Kolanski, R., and Heiser, G. (2014). Comprehensive formal verification of an os microkernel. *ACM Transactions on Computer Systems (TOCS)*, 32(1):2.

Klein, G., Andronick, J., Fernandez, M., Kuz, I., Murray, T., and Heiser, G. (2018). Formally verified software in the real world. *Communications of the ACM*, 61(10):68–77.

Klein, G., Elphinstone, K., Heiser, G., Andronick, J., Cock, D., Derrin, P., Elkaduwe, D., Engelhardt, K., Kolanski, R., Norrish, M., et al. (2009). sel4: Formal verification of an os kernel. In *Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles*, pages 207–220. ACM.

Krohn, M. and Tromer, E. (2009). Noninterference for a practical difc-based operating system. In *30th IEEE Symposium on Security and Privacy*, pages 61–76. IEEE.

Kumar, N. N. and Shyamasundar, R. (2014). Realizing purpose-based privacy policies succinctly via information-flow labels. In *Proceedings of IEEE Fourth International Conference on Big Data and Cloud Computing (BdCloud 2014)*, pages 753–760. IEEE.

Kumar, N. N. and Shyamasundar, R. (2017). A complete generative label model for lattice-based access control models. In *International Conference on Software Engineering and Formal Methods*, pages 35–53. Springer.

Kuz, I., Liu, Y., Gorton, I., and Heiser, G. (2007). Camkes: A component model for secure microkernel-based embedded systems. *Journal of Systems and Software*, 80(5):687–699.

Lampson, B. (2011). Technical perspective making untrusted code useful. *Communications of the ACM*, 54(11).

Loscocco, P. and Smalley, S. (2001). Integrating flexible support for security policies into the linux operating system. In *USENIX Annual Technical Conference, FREENIX Track*, pages 29–42.

Martin, W., White, P., Taylor, F., and Goldberg, A. (2000). Formal construction of the mathematically analyzed separation kernel. In *Proceedings of Fifteenth IEEE International Conference on Automated Software Engineering (ASE 2000)*, pages 133–141. IEEE.

Murray, T., Matichuk, D., Brassil, M., Gammie, P., Bourke, T., Seefried, S., Lewis, C., Gao, X., and Klein, G. (2013). sel4: from general purpose to a proof of information flow enforcement. In *2013 IEEE Symposium on Security and Privacy*, pages 415–429. IEEE.

Richards, R. J. (2010). Modeling and security analysis of a commercial real-time operating system kernel. In *Design and Verification of Microprocessor Systems for High-Assurance Applications*, pages 301–322. Springer.

Sewell, T., Winwood, S., Gammie, P., Murray, T., Andronick, J., and Klein, G. (2011). sel4 enforces integrity. In *International Conference on Interactive Theorem Proving*, pages 325–340. Springer.

Smalley, S., Vance, C., and Salamon, W. (2001). Implementing selinux as a linux security module. *NAI Labs Report*, 1(43):139.

Walker, B. J., Kemmerer, R. A., and Popek, G. J. (1980). Specification and verification of the ucla unix security kernel. *Communications of the ACM*, 23(2):118–131.

Zeldovich, N., Boyd-Wickizer, S., Kohler, E., and Mazières, D. (2011). Making information flow explicit in histar. *Communications of the ACM*, 54(11):93–101.