OS VERIFICATION- A SURVEY AS A SOURCE OF FUTURE CHALLENGES

Kushal Anjaria and Arun Mishra

Department of Computer Science &Engineering, DIAT, Pune, India

ABSTRACT

Formal verification of an operating system kernel manifests absence of errors in the kernel and establishes trust in it. This paper evaluates various projects on operating system kernel verification and presents in-depth survey of them. The methodologies and contributions of operating system verification projects have been discussed in the present work. At the end, few unattended and interesting future challenges in operating system verification area have been discussed and possible directions towards the challenge solution have been described in brief.

KEYWORDS

Formal verification, operating system, kernel.

1. INTRODUCTION

The security and reliability of computer system is dependent on the underlying operating system kernel; kernel is the core of operating system. The kernel provide mechanism for user level applications to access hardware, scheduling and inter-process communication. Therefore, if anything goes wrong in the kernel while programming or implementation, it will affect the operation of entire system. To ensure the correct working of a kernel, testing and/or verification techniques have been used. Testing reduces frequency of failures, while verification detects errors and eliminates failures. Consider the fragment of C code shown in figure-1

```
int div(int p, int q)
{
    int r;
    if (p<q)
    r=p/q;
    else
    r=q/p;
    return r;
}
```

Figure1. Fragment of C code

Here r being an integer, if testing has been performed with p=8, q=4 and p=16, q=32, full coverage of code can be achieved. Full coverage implies each line of code is executed at least once and every condition has been tested once. But here still two divide by zero errors are remaining. To identify these errors, human tester will immediately suggest that p=0, q=-1 and p=-1, q=0 should be tested. But for bigger and non-trivial program, it is difficult to find these cases and tester can never be sure that all such cases are covered.

From the example similar to above, G. Klein[19] has concluded that humans are good at creativity, they are not so good at repetitive, complex and high detailed task. But operating system verification is complex and needs many repetitive tasks. That's why formal verification of operating system is feasible compared to normal testing. Formal verification of an operating system kernel produces mathematical proofs of correctness of an operating system. Formal verification may differ from the user's view of correctness, this difference is called semantic gap[51]. Bridging this semantic gap is called formalisation[52]. The figure-2 below shows the entire verification procedure of any system with formalisation. The specification block in the figure-2 describes the collection of mathematical entities; these entities will be in the form which can be analysed by mathematical methods later. The model block describes mathematical representation of the system at some chosen level of abstraction. The verification tool will take mathematical model and mathematical entities as input and it will verify that the model is correct for all the entities or not. After that it will generate the verification result.

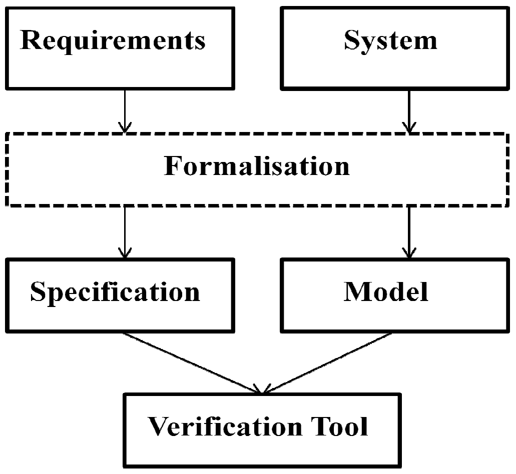

Figure 2. Verification procedure [52]

After brief overview of the verification procedure, some of the formal verification projects of operating system have been surveyed. Before moving on to the operating system verification survey, some basic terminologies related to OS verification have been explained:

a. Model checking: Model checking is formal verification method for finite state concurrent system. Large systems, like OS can reside in many states. That means they have large state space. Model checking can be applied to abstract model of real systems only. This diminution restricts the conclusion which can be drawn from model checking for operating system[27, 55, 56].
b. Proof-carrying code: Proof carrying code is an approach in which the kernel accepts only those extensions which are accompanied with valid proof for particular security policy. It tackles the problem of untrusted code execution in kernel mode[27, 57, 58].
c. Static source-code checking: Static source-code checking statically performs the analysis of source code. The static source-code checking analyzes the source code and gives guarantees about the absence of errors, while testing runs the system for code analysis. In this way it is different from testing[27, 59].
d. Functional correctness: The functional correctness in OS kernel verification means that the implementation always strictly follows high-level abstract specification of kernel behaviour[43]. Functional correctness makes it feasible to prove security properties at the code level. It does not necessarily imply the security. Functional correctness provides reasons about an implementation and a specification. Functional correctness proof has been considered as the right first step and basis for proving high level properties[27, 60, 61] .

2. An overview of operating system verification projects

To capture the wide domain of operating system verification, following projects have been surveyed. In the present survey UCLA, KIT, PSOS, VFiasco, EROS and seL4 projects have been discussed. These projects are among the prominent OS verification projects and it has been believed in the present work that survey of these projects is sufficient to find new challenges in OS verification field.

2.1. UCLA Project

The verification of UCLA system is based on data security model provided by G. Popek and D. Farber[1]. The data security model can be used to verify many of those properties in an operating system which are necessary to ensure reliable security enforcement. They haven't tried to prove that an operating system is entirely correct; instead they centralized all the operations which affect the security into nucleus. This nucleus is called the security kernel. If the operations of this kernel are correct then this implies that entire system is secure. Besides UCLA, the application of this approach has been used in other areas also, e.g. to design security model for military message systems[2] and to decide dependability of trusted bases[3].

UCLA secure unix[4] and Provably secure operating system (PSOS)[15, 16] have been considered as first serious attempts to verify an OS kernel. These projects were attempted 35 years ago. The UCLA secure unix had been developed as an operating system for the DEC PDP-11/45 computer. The project tried formal modelling and verification of Unix kernel, which was written in simplified pascal.

Project Implementation

The project was divided into two parts: first, a four level specification, ranging from Pascal code at bottom to top level specification was developed. Then in verification, it needs to be proved that different levels of abstractions were consistent with each other. The UCLA project managed to finish 90% of their specification and 20% of their proofs in 5 person-year.

Figure-3 shows the detail of implementation of UCLA secure unix project. Left hand side figure shows the specification layer, used in this project. All the specifications in the specification layers are called state machine. Here instead of one specification layer, multiple specification layers were designed because proof consistency can be handled easily with the multiple layers. The Pascal code block is actual Pascal code of kernel. The low level specification block in figure-3 describes data structures of implementation, in which some of the details are omitted. The abstract level contains specific objects like process, pages and devices. The top level specification described in figure-3 actually contains data security notion. This data security notion has been discussed by G. Popek and D. Farber[1]. The figure in right shows the consistency proofs between the levels. These proofs show that the specifications are consistent with each other. The proofs define functions. The function maps program state of concrete level to the program state of abstract level. The figure in right describes that for each operation in concrete system, corresponding operation in abstract system transforms the mapped concrete state accordingly [8]. Now the points below show some important findings that can be derived from UCLA project, which will connect us with the current formal verification scenario:

- Prior efforts to this project, to make operating system secure merely found the flaws in the system, so it became clear that piecemeal alterations were unlikely ever to succeed [6]. So, this UCLA project has been seen as a systematic formal approach that control OS's design and implementation.

- In UCLA they first generate the nucleus using the approach described in [1]. This nucleus was very close to modern microkernels[27], although at that time microkernels hadn't been invented[5].

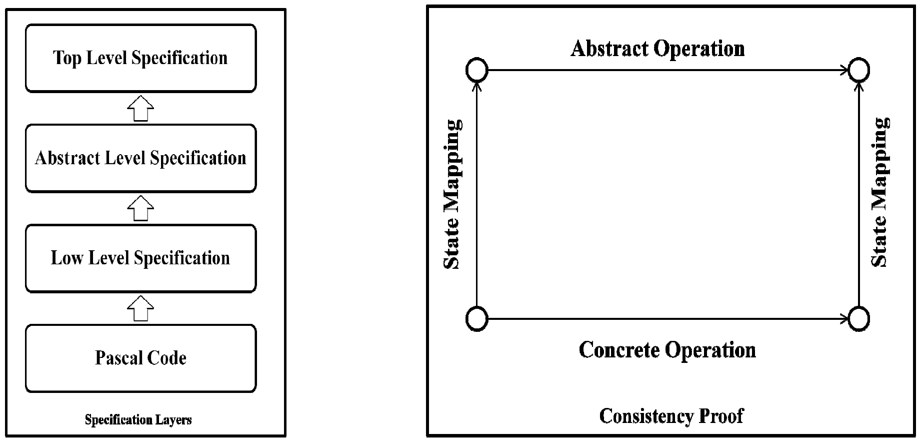

Figure 3. UCLA specification layers and consistency proofs [8]

- It has been discussed by the authors that the error has been discovered which justifies the need of formal specification. The error was related to boundary condition of kernel call. Generally boundary condition of kernel call maps a page into user-address space hadn't been properly handled. Mischievous process can read and read or modify the memory pages adjacent to its own.
- When the project took place, formal refinement hadn't came into existence, though the technique used in this project is formal refinement, defined by Morgan[7]. In fact, it is data refinement technique[8].
- At the end of this project, it has been observed that performance of the system was poor, it was slower than the standard unix in some cases. This was because of the nature of the pascal compiler and high context switch cost. Modern microkernels have overcome this limitation. For example SeL4 kernel. The Sel4 project has been described later in this paper.
- Authors have observed that the approach of program verification and proof development before or during the software development is not practical. They have also said that if system needs to be verified then it should be developed with verification in mind. The similar conclusion has been drawn from other projects as well for example SeL4[43,44,45] project.

2.2. KIT Project

KIT(Kernel for Isolated Task)[9] was the first operating system which had been verified at the assembly level. Main target of multitasking operating system is to implement the processes. The purpose of KIT project was to verify whether all these processes are isolated or not. Name KIT stands for 'Kernel for Isolated Task', that means task can communicate only in specified way. KIT is written in the machine language of uniprocessor von-Neumann computer. KIT kernel provides exception handling, access to asynchronous I/O devices and a single word message passing. KIT kernel had been implemented in artificial, yet realistic assembly language. Its code size was 620 lines, which was very small. Out of these 620 lines only 300 lines were of actual instructions [5].

Project Implementation

The KIT kernel has been formalized in Boyer-Moore logic[10] and all the proofs have been checked mechanically by Boyer-Moore theorem prover[10]. KIT verification has been done by proving correspondence between the two finite state machines' behaviour: abstract kernel finite state machine and target machine finite state machine. To describe finite state machines, description of the set of machine states and a definition of each transition on a machine state are required.

Now the points below show some important findings that can be derived from KIT project:

- There is a fiction in KIT OS that each process owns the processor. The processor state maintains this fiction. The verification of KIT proves that the processor state has been saved correctly.
- An interpreter equivalence theorem establishes implementation relation. This relation is similar to the Milner's weak simulation relation [11].
- KIT described process isolation properties down to object code level, but it was simpler and had less general abstraction than modern microkernel [12]. But in terms of verification, KIT had same order of complexity as modern microkernel [13].
- KIT was different from current microkernels because, it doesn't provide dynamic process creation, virtual memory in modern sense and no support for shared memory [5].

2.3. PSOS

PSOS (Provably Secure Operating System)[15,16] project was based on Robinson–Levitt paper[14], which had introduced the concept of formal mappings between different level of functional specifications that represented abstract implementations of each layer as a function of the lower layers. This layered system architecture can be seen in figure-4 below. The interface between each layer serves as a high level functional specification for lower layer and at the same time it serves as a machine model for the higher layer.

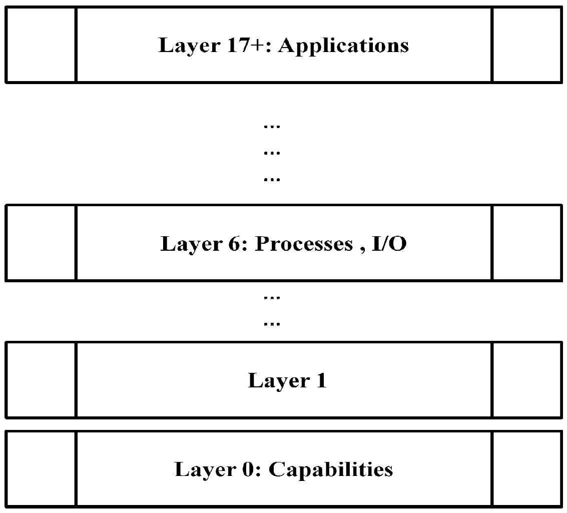

Figure 4. PSOS layered system architecture [8]

Project implementation

As shown in the figure-4, PSOS has 17 layers in its layered system architecture. Out of these 17 layers, bottom six layers are implemented using hardware and layers seven to seventeen are implemented using software. As shown in the figure, layer 0 is tagged. Tagged means it has hardware enforced capabilities; hardware supports a bit that indicates whether a word in memory stores a capability or not. This layered architecture is similar to the TCP/IP network stack, that's why in the figure it has been shown that the top layer contains application. Similar to the TCP/IP network stack, the top layer contains all the applications which have been needed by the layers below it.

It had been believed in the project that layering would make the formal verification easier. PSOS was a capability based system in which incorporated hardware implemented capability. Now the points below show some important findings that can be derived from PSOS project:

- Hardware implemented capability in PSOS had weakened the primitive similar in some respects to the diminish operator; existing access rights can be selectively retained. It is unclear from the literature whether the application of this operation was imposed by an access right or was discretionary [17].
- PSOS also implemented concept named "store permissions". This mechanism could selectively control which capabilities can be stored to which capability segments. This feature can be used to enforce write down permissions [17].
- The properties of application specific object type could enforce with the hardware assistance provided with the capability based access control. The design allowed application layers to efficiently execute instructions, with object-oriented capability-based addressing directly to the hardware, although it appeared at a much higher level of abstraction in design specification [18].

2.4. VFiasco project

The main challenge in VFiasco project [27] is to entitle high level reasoning in terms of typed objects during verification, yet assume only low level hardware properties. The VFiasco project aims at mechanical verification of security relevant properties of Fiasco microkernel. Fiasco microkernel is L4-compatible Fiasco microkernel [28]. The aim of the project is an OS kernel which provides security guarantees which has been verified.

Project implementation

Fiasco kernel has been implemented in C++. The language C++ is not the language with precise semantics. For this purpose, M. Hohmuth et al.[27] developed a language which had precise semantics called 'safe C++'. After converting code to safe C++, the verification will be carried out in the theorem prover Isabelle/HOL [29]. In this project, conversion from safe C++ to HOL semantics has been done automatically by the logic compiler. For verification, authors have abstraction level of virtual machine that provides a type-safe object store which is a memory that supports reading and writing of typed values.

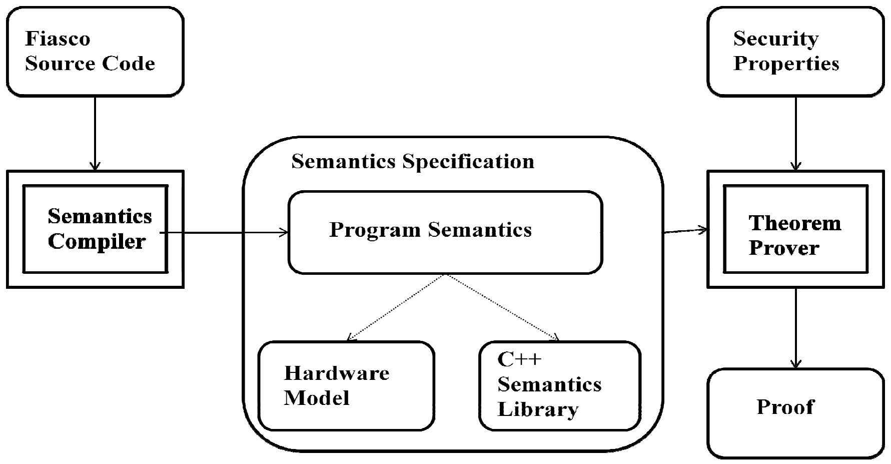

Figure 5. The VFiasco project verification overview [40]

In this project the existence of an object store layer with strong properties is a proof not an assumption. The approach used in this project to a semantics of C++ is very close to the one used in LOOP project[30] of Java. Figure-5 below gives an overview of VFiasco verification process. First semantic compiler will translate C++ code into semantics, formulated in higher order logic. Then hardware model and C++ semantic library provide functions to express program semantics. After that theorem prover verifies the semantic specification against security properties. Finally verification results in proof.

Now the points below show some important findings that can be derived from VFiasco project, which will connect us with the current formal verification scenario:

- The VFiasco project uses coalgebraic[31] methods. It describes coalgebraic class specification language called CCSL. Coalgebraic proof methods are not only characteristic capturing formalism for non-terminating program, but also used for labelled transition system[32].
- In VFiasco project source code verification has been directly applied to the unmodified source of Fiasco microkernel operating system written in C++. In C++ it is not possible to jump across the function boundaries. So formal reconstruction of goto loop, which has been described in [33] needs to be applied. For complete C++ semantics one needs the semantics for data-types that can deal with the type cast of data and of pointers. During verification, state transformation approach has been used to get relatively simple semantics to statements like break, continue and even goto[27]. The state transformation has been explained by M. Huisman and B. Jacob [30].
- The VFiasco project contains single layer, i.e. object store layer. This layer provides functions for typed objects, in such a way that typed objects can be safely manipulated. Because of this single layer, The VFiasco project resides in ‘single-layer approach’ category (ShengWen Gong [34] has classified verification efforts/projects into four categories. These four categories are: (i) the component approach [35], (ii) the parallel approach[36], (iii) the single-layer approach[27], (iv) the pervasive approach[37]).
- The VFiasco project stopped at the source-code level. That means, there are no attempts to map results down to lower systems or language layers. This project doesn’t involve any compiler verification [38].

2.5. EROS project

The EROS (Extremely Reliable Operating System)[47,48] is a capability based operating system. It is for commodity processors which uses a single level storage model. J. Shapiro et al. have formalized and analyzed the security model of EROS using the pen and paper approach. The security model was implemented based on take-grant model[42] of capability distribution. The security model of the EROS was not formally connected to the implementation.

Project implementation

The EROS project actually started by verifying confinement mechanism. J. S. Shapiro and S. Weber [48] provided formal definition of confinement policy in their work. After that they have explained operations and security requirements of real operating system. In their work, methodology and proof structures have been developed for confinement policy in capability based structure. They have claimed that their methodology can be generalized to solve information flow problems in many capability based architecture.

Now the points below show some important findings that can be derived from EROS project:

- One thing about EROS is, when machine starts, it loads a fully pre-initialised state. This task makes it possible to inspect the initialised state offline[49].
- S. Maffeis et al.[50] provide the object capability based model that provides approach for isolating untrusted components in web applications. Their work is close to EROS confinement mechanism. While there are some other similarities between their framework and our general setup, one substantial difference is that instead of defining authority as an over-approximation of heap actions that can be performed by a single object; they define authority for the whole system[50].
- The Coyotos kernel[64, 65] was successor to the EROS kernel. From the security model of EROS kernel, Shapiro et al.[54] concluded that there should be a formal connection between security model of the kernel and the implementation. They have tried to establish this formal connection in Coyotos kernel[8].
- The EROS system currently runs on Pentium hardware. Future details about the project can be obtained from its website[62].
- The CapROS (Capability based Reliable Operating System)[63] project is the continuation project of EROS. It is using same EROS code base. The CapROS project is being led by Charles Landau.

2.6. seL4 verification project

The seL4 (Secure Embedded L4) kernel is an evolution of the L4 microkernel. It is third generation microkernel of L4 provenance. It targets embedded devices. The seL4 implements capability based protection system, capabilities in seL4 are immutable. Similar to seL4 micro kernel, seL4 provides address spaces, inter-process communication and threads. In seL4 all system calls are invocations of capabilities. The seL4 comprises of 8,700 lines of C code and 600 lines of assembler.

Project implementation

The seL4 verification project[43,44,45] was the project which provided the proof of functional correctness of a complete general purpose operating system for the first time. In this project, formal machine-checked verification of seL4 microkernel has been performed from an abstract specification down to the C implementation. In this project, first the access control model has been verified and then the actual functional verification of kernel had been started. Before the project, correctness of compiler, assembly code, boot code, management of caches and hardware has been assumed.

An access control model of seL4 has been verified by D. Elkaduwe et al.[41] in the theorem prover isabelle/HOL[29]. In their work the take-grant[42] model has been used, but without take rule and with more realistic create rule that is explicitly authorized by capability. In their formalization, remove rule has also been modified. In seL4, remove-rule will remove whole capability instead of removing few parts. D. Elkaduwe et al.[41] have proved that kernel provided mechanisms are sufficient to enforce mandatory isolation between subsystems. They have shown that it is possible to build fully spatially separated system on top of seL4. So here spatial memory separation has been guaranteed but authors have said that with current stock hardware preventing all covert timing channels is not possible.

The seL4 kernel design process has been shown in the figure-6 below. The square boxes show the formal artefacts. These artefacts have direct role in the proof. The double arrows shown in the figure represent implementation effort; the single arrows represent design influence of artefacts on other artefacts. The central artefact is the actual Haskell prototype of the kernel. The prototype requires the design and implementation of algorithms that manage the low-level hardware details. The figure-6 shows that hardware and Haskell prototype has design influence on formal executable specification.

After the design process, verification process took place. In this project, the verification was interactive, machine assisted and machine checked proof. Figure-7 shows the specification layers which were used in verification of seL4. Abstract specification layer shows the details to specify outer interface of the kernel. It doesn't describe in detail how the effects or interfaces are implemented in kernel; in short it describes what the system does without saying how it is done. The executable specification layer has been generated from Haskell into the theorem Prover. It contains all data structure and implementation details which have been expected in the final C implementation. The high performance C implementation layer deals with the formally-defined semantics.

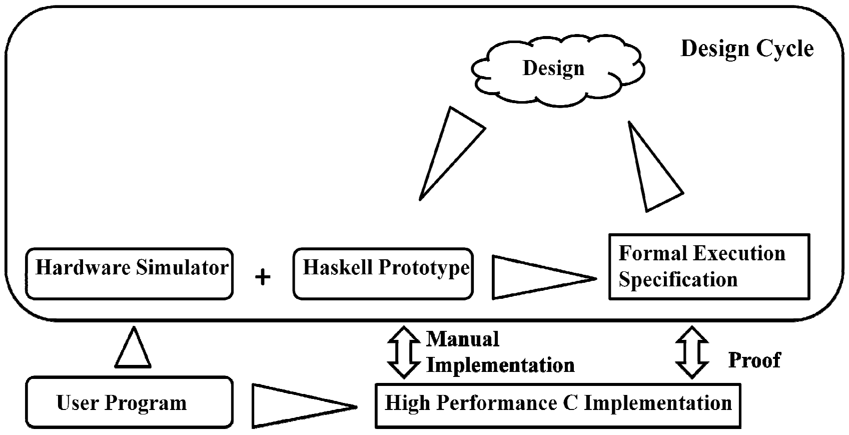

Figure 6. The seL4 design process [43]

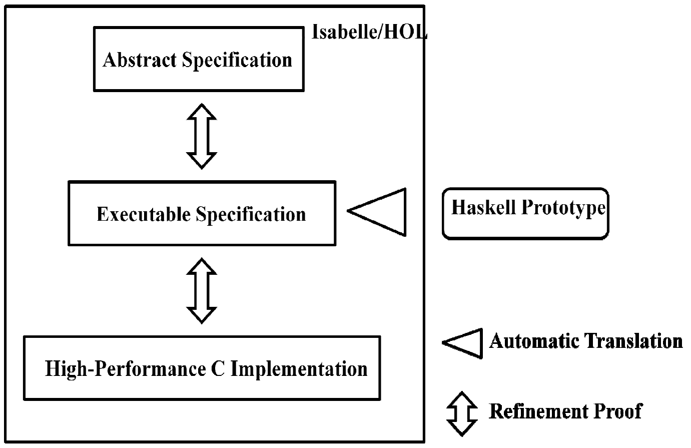

Figure 7. The refinement layers in the verification of seL4 [43]

Now the points below show some important findings that can be derived from seL4 project, which will connect us with the current formal verification scenario:

- There are many techniques for formal verification like model checking, static analysis or kernel implementations in type safe language. But in this project it has been believed that functional correctness is stronger and more precise technique compared to the techniques mentioned above[43].
- In this project, fusion of traditional operating system and formal method technique had been used i.e. rapid kernel design and implementation had been used together. Because of this implementation, the verification focus was improved and design was not conflicting with the better performance[43].
- In UCLA it had been observed that simplification of kernel to make verification feasible made the kernel little bit slower. But in this project it has been shown that with modern tools and technique, this is not the case these days.

The summary of the aforementioned projects has been shown in the table-1 below. In the year column, the question mark in parenthesis shows that the year is not known and year in parenthesis shows estimated completion date. Further, many verification projects like bias variance tradeoffs[66], the Xtratum[67], L4Android[68], ORIENTAIS[69], VTOS[70] and CHERI[71] have been studied. They are evolved from the aforementioned projects and follow same path as described in above projects for verification of OS.

Table 1. OS verification projects [8]

Project	**Highest level**	**Lowest level**	**Specs**	**Proofs**	**Prover**	**Approach**	**Year**
UCLA	Security model	Pascal	90%	20%	XIVUS	Alphard	(?)-1980
KIT	Isolated task	Assembly	100%	100%	Boyer Moore	Interpreter equivalence	(?)-1987
PSOS	Applicatio n level	Secure code	17 layers	0%	SPECI AL	HDM	1973-1983
VFiasco	Doesn't crash	C++	70%	0%	PVS	Semantic compiler	2001-2008

EROS	Security model	BitC	Security model	0%	ACL2(?)	Language based	2004-(?)
L4 verified	Security model	C/assembly	100%	70%	Isabelle	Performance production code	2005-(2008)

3. FUTURE CHALLENGES

Further, challenges out of the above projects and inherited projects have been discussed. In the present survey it has not been claimed that the challenges described in this section haven't been implemented at all. But to the best of our knowledge, the areas described in this section haven't been covered in detail in literatures.

From all the above projects, especially from the SeL4 project it can be concluded that functional correctness is one of the strongest properties that can be proven about the system. The functional correctness can help to make precise formal prediction of how the kernel behaves in all possible situations for all possible inputs. G. Klein[19] has suggested that, if any specific property is needed to be checked then it can be expressed in Hoare's logic, it is enough to work with this formal prediction, with the specification. But functional correctness property of any system can't prove that the system is secure, it just says that system is functionally correct [19]. The word 'secure' in secure system requires formal definition, but this depends on what you want to use the kernel for. So verification of all security properties or secure system which has been built on top of the OS kernel is one of the recent challenges in formal verification field.

In SeL4 project specific security properties hadn't been proven, but G. Klein[19] believed that if the functional correctness property of the operating system can be proven, then below assumptions can be made easily about that OS: (1) No code injection attacks (2) No buffer overflow attack (3) No NULL pointer access (4) No ill-typed pointer access (5) No memory leaks (6) No non-termination (7) No arithmetic or other exceptions (8) No unchecked user arguments. He has explained these assumptions with reasons. He has also explained that the functional correctness can tell following properties about the code: (1) Aligned object (2) Well formed data structures (3) Algorithmic invariants and (4) Correct book-keeping. So research challenges here are: can functional correctness property be used to verify other security properties or can framework be made that covers verification of most of the security properties.

G. Heiser et al.[20] have provided some research challenges in different way, they have told that the users of the fully verified kernel have trustworthy foundation for the entire system. For example, the kernel seL4 is fully verified and it can be used as trustworthy foundation. Now the challenge is how the trustworthy foundation of the system can be used further. They have suggested three uses of the trustworthy foundation: (1) Secure web browsing (2) Increase the usefulness of TPM (3) Cost-reduced database. But still there are many challenges related to the kernel used as trustworthy foundation.

Let's start with the secure web browsing topic. One security policy of the browser is Same Origin Policy (SOP). SOP means web pages from different sources cannot observe or alter each other's state and behaviour and script running inside the web page must denied unauthorized access to OS resources. Recent browsers like Chrome[22] address this problem by encapsulating security policy in separate module, the browser kernel. So, this approach is based on the OS, specifically dependent on browser's TCB. So one challenge here is can TCB of the browser be reduced using microkernel approach. IBOS[21] project has actually shown that TCB of browser can be reduced by using the microkernel approach. The authors have proposed architecture for secure browsing which contains two trusted parts: microkernel and user level security process which are shown in figure-8.

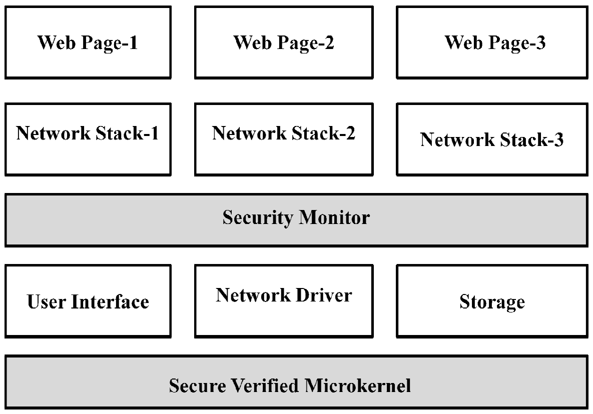

Figure 8. Secure web browser architecture. Components that belong to the TCB of the system are highlighted using grey shading[20]

Figure-8 shows that monitor is the only part in the TCB that needs to be verified. The monitor instantiates browser process with permissions to directly communicate with their private network, stack processes and the monitor itself[20]. Once this security monitor is verified one can have completely verified TCB for secure browsing. So, research challenge here is can the aforementioned architecture be extended to include complete OS stack and that too running inside a VM (Virtual Machine).

Next challenge is to increase usefulness of TPM. Trusted Computing Group[23] has introduced the Trusted Platform Module (TPM). The TPM provides remote attestation facility which provides evidence that the trusted software stack has been loaded by the remote system from its booting time. Now let’s take classical example of bank transaction. The bank can reject the transaction request from the remote system if the trusted software stack hasn’t been loaded by the remote machine. Bank clients want to use the smart phone or computer for the transaction. The smart phones and computers have many apps and software for the bank to manage. Many of these apps and software are not trusted. So, for successful remote attestation it has become necessary to kept these out of TCB and therefore the system. To solve this problem, the concept of Dynamic Root of Trust Measurement (DRTM)[24] has been introduced. DRTM allows user to switch between trusted and untrusted environment. But to achieve this trusted code should be small and it can run only for a fraction of seconds. But bank transaction needs more time than few seconds. So above approach needs user to suspend OS while he is doing the bank transaction. But this is not a practical solution. So to solve this issue TCB is needed that do not change, change rarely and change in controlled fashion. After concluding this G. Heiser et al. believed that formally verified kernel do not change or changes very rarely because it won’t require any bug fixes. They have given example of seL4 kernel also. The challenge here is to minimize TCB and isolate TCB from rest of the general purpose OS components using the trusted system which contains fully verified kernel.

Final challenge that G. Heiser et al.[20] have mentioned is about the database systems. The databases can have disk failures, power failures and OS crash[20]. These days RAID protects database from disk failure and UPS protects from power failure. This leaves protection against OS crash as one of the open research problem. Here research questions can be formed: can properties of kernel be verified in such a way that it can make kernel crash-proof, Can database be directly implemented on verified kernel and if it is possible then what kind of changes required in the lower layers of DBMS. Can database be implemented on verified kernel in such a way that the changes required in the lower layer of database become minimal and practical to achieve.

J. Andronick et al.[25] have tried to solve a research problem related to scalability of the formal verification. They have dealt with the large and complex system which uses seL4 for which security guarantees can be given. They have proposed a framework to build such large and complex systems. Authors' vision here is that not all the software in a large system necessarily contributes to a security property of interest. From the vision, the methodology has been developed: (1) isolate the software parts which are not critical to a targeted property and prove that for specific property nothing more is needed to be proven about them (2) formally verify and prove that remaining part satisfy the targeted security property. In this paper the case study of Secure Access Controller (SAC) has been taken. Access control-based security policy has been taken as a property of interest here. Authors have explained that verifying such a property for large system is far beyond the abilities of current verification methods. To verify this property large code base has been divided into trusted code and untrusted code. The authors have used here seL4[26] kernel to get the code isolation in this system which is already verified. So question arises that if the kernel which hasn't been verified is used for the large and complex system, can we get security for such system and get the code separation.

In 2001, during VFiasco project[27] M. Hohmuth et al. Observed that huge bug affected monolithic kernels are outside the scope of verification technology which were available at that time. Although the microkernels are smart choice for constructing verified secure system, but is it possible to have verification technology now which can accommodate verification of monolithic kernel. Verification of monolithic kernel can be costly in terms of time and efforts, but it can be taken as a recent research challenge.

The formal verification community has nice security properties, high level formal model and ways of architecting secure systems, but still no signs of implementation level proofs. Even recently implemented seL4 microkernel doesn't have these implementation level proofs. G. Klein et al.[39] believes that this needs to be changed. It is obvious that if the system is large then it can be secured by reducing the Trusted Computing Base (TCB). For large systems, microkernels provide good foundation, though with reduced TCB of large systems, nobody has proved security down to the implementation level. With type 1 hypervisors, it has been assumed that they will perform the role of separation kernel, but there are no implementation level proofs for these hypervisors either[39]. For a moment let's assume that in next few years, kernels are available with fully formally verified functional correctness down to the implementation level. But then what next? Such kernels do not automatically imply security [19].

After the completion of sel4 verification, G. Klein et al.[39] have mentioned in their work that the mandatory access control can be implemented on OSes for example in Linux, but it is not possible to get provable or assurable security for such OSes at least for next few years. Assume that all security properties of kernel have been proved in next few years (Sorry, this will never happen), still we as a formal verification community are not done yet. Our ultimate goal should be to achieve proof that whole systems enforce their security goals; we can get much stronger assurance for much larger systems than what is thought feasible today. It should be remembered that proofs say that the code will follow specification; it doesn't say that specification enforces specific security property.

From the projects like VFiasco and seL4, it can be observed that there is a gap between idealized security properties and properties that hold real kernels. G. Klein et al.[39] have mentioned in their work that there is no good formal handle available on timing and time based covert channels for practical implementations, so it is a research challenge. The authors are not expecting that the same level formal proofs will be obtained in near future as it is available for storage channel.

K Elphinstone[46] has described challenge that takes place while using verified microkernel like seL4 in Digital Rights Management (DRM), and based on this he has provided some new issues which can be addressed by research communities. One can think solutions of this problem in any direction, not necessarily in formal verification point of view. The DRM is the concept of specifying, enforcing and limiting rights associated with digital content. Before identifying issues, the architecture of DRM has been explained. The DRM architecture has been shown in the figure-9[46]. First user posses the device upon which he wishes to view content. Then the user must be authenticated by the content provider.

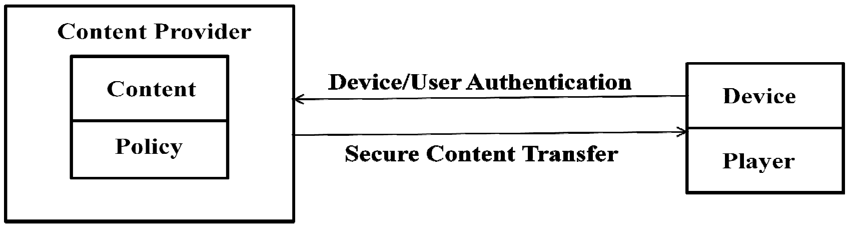

Figure 9. General DRM architecture [46]

The content must be securely transferred to user device, for this encryption has been used. The user player should be able to decrypt the content when user wishes to view it.

Here the content-use policy can be violated by end-users in many ways, ranging from reverse engineering, modifying players or running the player on modified operating system. This is the research challenge. One solution is to provide assurance that a trusted player on a trusted operating system is the only software that has access to the content, other solutions can be thought of in the context of hardware or networking.

Next, the challenge is about the gap between formal model of OS and hardware implementation. A. Cohn[51] has described that the physical hardware is a realisation of a model, and correct hardware operations are beyond the scope of the formal verification. One example of this is: manufacturers can't prove absence of manufacturing defects. So, even if the verified processor or kernel is available, the gap between formal model of such kernel or processor and implementation will always exist[51]. The research challenge is how to minimize this gap.

Tuch H. et al.[52] observed UCLA, KIT and VFiasco closely. They have described in their work that in kernel verification, challenges related to performance, size and the level of abstraction is still available. They have mentioned that features like direct hardware access, pointer arithmetic and embedded assembly code haven't been the subjects of mainstream verification research, so research scope is available in these areas.

The KIT was multitasking OS, which gave direction to the present survey to find verification challenges in concurrent OS. S. Rajamani et al.[53] have discussed challenges of an OS which support concurrent execution of the program. Suppose an OS which supports concurrent execution of program has normal page size. This OS supports third party plug-ins. In this type of OS, there are many research challenges: (1) Is it possible to guarantee isolation in this OS where third party plug-ins can be ill-behaved[53]? (2) With the isolation guarantee in OS, is it possible to manage properties like safety and permissiveness? (3) Is it possible to achieve isolation without the problems like deadlocks, livelocks[53], memory fragmentation and condition variable handling? (4) Various operating systems provide various types of memory protections. How to handle these variations if memory protection is used for isolation?

The challenges are summarized in a table below:

Table 2. Challenges in formal verification

Challenge	**Explanation**	**Source**	**Suggestion**
Functional correctness	Although functional correctness is strong property, it can't prove that system is secure. To verify more number of security properties together with or without the functional correctness is a challenge.	seL4 project	R. Akella and Bruce M.[72] have verified security properties like information flow and non deducibility of cyber-physical system. They have described that information flow security can't be checked using functional correctness. So, they have provided different framework for verification using process algebra. Same methodologies can be applied to OS verification also.
Kernel as a trustworthy foundation	Three uses of the trustworthy foundation: (1) Secure web browsing (2) Increase the usefulness of TPM (3) Cost-reduced database. The challenge is how the usage of trustworthy foundation in different areas can be increased.	seL4 project	G. Heiser et al.[73] have provided prototype system RapiLog which is based on verified seL4 hypervisors. This system is used to reduce system complexity by leveraging verification instead of using special hardware. Similarly more ways can be found where kernel can be used as trustworthy foundation.
Security in large system	Not all software on the large system contributes to the security of system. So to verify the large system, system should be divided in trusted and untrusted code. The division of large system in trusted and untrusted code is a challenge.	VFiasco, seL4	Stefan et al[79]. have verified cyber-physical system. Their work provides hint to verify larger and more complex systems.
Implementation level proof	No implementation level proofs so far for any kernel verification. This is a challenge.	Recent projects like seL4, VFiasco	Nana S. et al.[74] have provided framework to verify hardware and software co-verification. IEEE 802.11ac WLAN system has been selected by them for case study. Their work might provide a hint to solve this research challenge.

Time-based covert channel	To create formal handle for time-based covert channel is a challenge.	Recent projects like seL4, VFiasco	The problem with covert channel is it is difficult to verify it on case-by-case bases. So, first unique model to unified approach is needed and then verification can be done. So P. L. Shrestha et al[78] have provided this unique model which can be used to verify time-based covert channel.
Monolithic kernel	Huge, bug affected monolithic kernel verification is a challenge.	VFiasco project	M. Lange et al[77]. have provided hint to solve this research challenge. Their work focuses on L4Android which is monolithic architecture. Although the project is not entirely on the monolithic kernel verification, but it can provide help to solve this research problem.
Challenges related to unexplored areas	Pointer arithmetic and embedded assembly code haven't been the subjects of mainstream verification research so far. These fields can be explored more.	UCLA, PSOS, KIT, VFiasco, EROS	The work of Thomas S. et al[75]. gives hint in the direction of pointer arithmetic. J Kobashi et al[76] have provided directions to verify embedded assembly code. Their work can be explored further to solve these challenges.
Challenges related to Concurrency	(1) To achieve and verify isolation when third party plug-ins are ill-behaved. (2) With isolation, manage the properties like safety and permissiveness (3) Avoid problems like deadlocks, livelocks, memory fragmentation and condition variable handling (4) Handle variations in memory protection if it has been used to achieve isolation.	KIT is multitasking kernel. It has provided base to explore concurrency related areas.	Solutions of some of the problem have been discussed by S. Rajamani et al[53]. They have tried to solve this challenge for concurrent programs. These solutions can be expanded for the large, concurrent operating system kernel and the isolation properties can be verified.

4. CONCLUSIONS

In the present paper, an overview of kernel verification projects has been provided, in specific UCLA, KIT, PSOS, VFiasco, EROS and seL4 projects have been surveyed. In the present survey, highest level specification, lowest level of specification, model checker and approach of verification have been described for each project. During the survey it has been observed that proofs of automation, memory models, proof libraries, and program logics have been developed significantly, that's why OS verification is not as hard as it was before 35 years.

The present article also describes challenges in kernel verification area. They are related to monolithic kernel verification, functional correctness, implementation level proof, time-based covert channel, direct hardware access, pointer arithmetic, concurrency and use of kernel as trustworthy foundation areas. Future efforts in the direction of solving these challenges will make verification faster and precise.

REFERENCES

[1] G. Popek and D. Farber, (1978) "A model for verification of data security in operating systems," Communications of the ACM, vol. 21,(9), pp. 737–749.

[2] Landwehr, C. E., Heitmeyer, C. L., and Mc Lean, J., (1984) "A security model for military message" systems. ACM Trans. Comput. Syst, 2, 3, 198–222.

[3] E. Kang and D. Jackson, (2010) Dependability arguments with trusted bases. In RE, pages 262-271. IEEE Computer Society.

[4] Walker, B. J., Kemmere, R. A., and Popek, G. J., (1979) "Specification and Verification of the UCLA Unix Security Kernel". In Proceedings of the 7th ACM Symposium on Operating Systems Principles (SOSP), pp. 64–65.

[5] T. In der Rieden, (2009) "Verified Linking for Modular Kernel Verification". PhD thesis, Saarland University, Computer Science Department.

[6] Popek, G.J., (1974) Protection Structures. Computer, 7(6), 22-33.

[7] C. C.Morgan, (1990) Programming from specifications. Prentice-Hall.

[8] G. Klein, (2009) "Operating system verification — an overview". Sadhana, 34(1):27–69.

[9] Bevier,W.R., (1989) "Kit: a study in operating system verification" IEEE Trans. Softw. Eng., 15(11), 1382–1396

[10] R. S. Boyer and J. S. Moore, (1988) A Computational Logic Handbook. New York: Academic.

[11] R. Milner, (1971) "An algebraic definition of simulation between programs", Stanford AI Project. Tech. Rep., AIM-142,.

[12] G. Heiser, K. Elphinstone, I. Kuz, G. Klein, and S. M. Petters, (2007) "Towards trustworthy computing systems: Taking microkernels to the next level" ACM Operating Systems Review, 41(3).

[13] J. S. Shapiro, M. S. Doerrie, E. Northup, S. Sridhar, and M. Miller (2004) "Towards a Verified, General-Purpose Operating System Kernel" In Proceedings of the 1st NICTA Workshop on Operating System Verification, pages 1–18.

[14] L. Robinson and K.N. Levitt, (1977) "Proof techniques for hierarchically structured programs", Communications of the ACM, 20(4):271–283.

[15] P.G. Neumann, R.S. Boyer, R.J. Feiertag, K.N. Levitt, and L. Robinson, (1980) "A Provably Secure Operating System: The system, its applications, and proofs" Technical report, Computer Science Laboratory, SRI International, Menlo Park, California, 2nd edition, Report CSL-116.

[16] P.G. Neumann and R.J. Feiertag, (2003) "PSOS revisited", In Proceedings of the 19th Annual Computer Security Applications Conference, Classic Papers section, pages 208–216,

[17] Jonathan S. Shapiro, (2003) "The practical application of a decidable access model" Technical Report SRL, November, Baltimore, MD 21218.

[18] R. N. Watson, P. G. Neumann, J. Woodruff, J. Anderson, D. Chisnall, B. Davis, B. Laurie, S. W. Moore, S. J. Murdoch, and M. Roe, (2014) "Capability Hardware Enhanced RISC Instructions: CHERI Instruction-set architecture", Technical Report UCAM-CL-TR-850, University of Cambridge, Computer Laboratory, Apr. URL

[19] G. Klein (2009) "Correct OS kernel? proof? done!", USENIX, 34(6):28–34.

[20] G. Heiser, L. Ryzhyk, M. von Tessin, and A. Budzynowski, (2011) "What if you could actually Trust your kernel?" In 13th HotOS, Napa, CA, USA,

[21] Tang, S., Mai, H., And King, S. T., (2010) "Trust and protection in the Illinois Browser Operating System", In 9th OSDI, Vancouver, Canada, pp. 1–15.

[22] Barth, A., Jackson, C., Reis, C., And The Google Chrome team, (2008) "The security architecture of the Chromium browser". Technical report, Stanford Security Laboratory.
[23] Trusted Computing Group. Trusted Platform Module. http://www.trustedcomputinggroup.org/developers/trustedplatform_module.
[24] McCune, J. M., Parno, B., Perrig, A., Reiter, M. K., AND Isozaki, H. (2008) "Flicker: An execution infrastructure for TCB minimization", In 3rd EuroSys Conf.
[25] June Andronick, David Greenaway, and Kevin Elphinstone, (2010) "Towards proving security in the presence of large untrusted components", In Gerwin Klein, Ralf Huuck, and Bastian Schlich, editors, Proceedings of the 5th Workshop on Systems Software Verification, Vancouver, Canada, USENIX.
[26] seL4Website. http://ertos.nicta.com.au/research/sel4/, Jun 2010.
[27] M. Hohmuth, H. Tews, and S. G. Stephens, (2002) "Applying source-code verification to a microkernel — the VFiasco project" (extended abstract) In Proceedings of the Tenth ACM SIGOPS European Workshop, September.
[28] M. Hohmuth and H. Härtig, (2001) "Pragmatic nonblocking synchronization for real-time systems" In USENIX Annual Technical Conference, Boston, MA.
[29] L. C. Paulson, (1994) Isabelle: A Generic Theorem Prover. Number 828 in LNCS. Springer, Berlin.
[30] M. Huisman and B. Jacobs, (2000) "Java program verification via a Hoare logic with abrupt termination", In T.Maibaum, editor, Fundamental Approaches to Software Engineering, number 1783 in LNCS.
[31] Jan Rothe, Hendrik Tews, and Bart Jacobs. (2001) "The Coalgebraic Class Specification Language CCSL", J. Univ. Comp. Sc., 7(2):175–193.
[32] Glesner, S., Leitner, J., Blech, J.O. (2006) "Coinductive verification of program optimizations using similarity relations", In Knoop, J., Necula, G. C., Zimmermann, W. (Eds.): Proc. of 5th Int. Wksh. on Compiler Optimization Meets Compiler Verification, COCV '06 Vienna,. Electron. Notes in Theor. Comput. Sci., Vol. 176(3), Elsevier pp. 61–77.
[33] Tews, Hendrik, (2004) "Verifying Duff's device".
[34] Gong, Sheng Wen, (2013) "Formal Model of Classic Operating System Kernel", Advanced Materials Research, pp. 1020- 1023.
[35] William R. Bevier, Richard Cohen, and Jeff Turner, (2013) "A specification for the Synergy file system", Technical Report 120. Computational Logic Inc.
[36] Hermann Hartig, Michael Hohmuth, Norman Feske, Christian Helmuth, Adam Lackorzynski, Frank Mehnert, and Michael Peter, (2005) "The Nizza secure-system architecture", Proc. of the 1st International Conference on Collaborative Computing: Networking, Applications and Worksharing.
[37] J. Strother Moore, (2002) "A grand challenge proposal for formal methods: A verified stack", Proc. of the 10th Anniversary Colloquium of UNU/IIST. pp. 161-172.
[38] Leinenbach, D., (2008) "Compiler Verification in the Context of Pervasive System Verification", PhD thesis, , Saarland University, Saarbrücken .
[39] G. Klein, T. Murray, P. Gammie, T. Sewell, and S. Winwood, (2011) "Provable security: How feasible is it?" In 13th HotOS, Napa, CA, USA, pp. 28–32.
[40] Matthias Daum, (2003) "Develoment of a Semantics Compiler for C++", Diploma thesis, TU Dresden.
[41] D. Elkaduwe, G. Klein, and K. Elphinstone, (2008) "Verified protection model of the seL4 microkernel", In J. Woodcock and N. Shankar, editors, VSTTE 2008 — Verified Softw.: Theories, Tools & Experiments, volume 5295 of LNCS, Springer , pp. 99–114.
[42] R. J. Lipton and L. Snyder, (1977) A linear time algorithm for deciding subject security, ACM, 24(3):455–464.
[43] G. Klein, K. Elphinstone, G. Heiser, J. Andronick, D. Cock, P. Derrin, D. Elkaduwe, K. Engelhardt, R. Kolanski, M. Norrish, T. Sewell, H. Tuch, and S. Winwood (2009) "seL4: Formal verification of an OS kernel" In Proceedings of the 22nd ACM Symposium on Operating Systems Principles.
[44] Gerwin Klein, June Andronick, Kevin Elphinstone, Gernot Heiser, David Cock, Philip Derrin, Dhammika Elkaduwe, Kai Engelhardt, Rafal Kolanski, Michael Norrish, Thomas Sewell, Harvey Tuch, and Simon Winwood, (2010) "seL4: formal verification of an operating-system kernel", 53(6), pp.107–115.
[45] G. Klein, P. Derrin, and K. Elphinstone, (2009) "Experience report: seL4 — formally verifying a high-performance microkernel". In 14th ICFP.
[46] Kevin Elphinstone, (2004) "Future directions in the evolution of the L4 microkernel", In Proceedings of the NICTA workshop on OS verification.

[47] Shapiro J S, Smith JM, Farber D J., (1999) "EROS: a fast capability system", In: SOSP 99: Proceedings of the seventeenth ACM symposium on Operating systems principles, New York, NY, USA: ACM pp. 170–185.
[48] Shapiro J S, Weber S, (2000) "Verifying the EROS confinement mechanism", In: Proceedings of the IEEE Symposium on Security and Privacy, IEEE Computer Society, Washington, DC, USA pp. 166–176.
[49] Andrew Boyton, June Andronick, Callum Bannister, Matthew Fernandez, Xin Gao, David Greenaway, Gerwin Klein, Corey Lewis, and Thomas Sewell, (2013) "Formally verified system initialisation", In Lindsay Groves and Jing Sun, editors, Proceedings of the 15th International Conference on Formal Engineering Methods, Queenstown, New Zealand, Springer pp. 70–85.
[50] S. Maffeis, J. C. Mitchell, and A. Taly, (2010) "Object Capabilities and Isolation of Untrusted Web Applications", In Proceedings of the IEEE Symposium on Security and Privacy, pp. 125–140.
[51] Cohn, (1989) "The notion of proof in hardware verification" Journal of Automated Reasoning, 5(2) pp. 127-139.
[52] Tuch H, Klein G, Heiser G. OS verification—now!. In: Proceedings of the 10th Workshop on Hot Topics in Operating Systems, 2005,USENIX, Santa Fe, NM, USA , pp. 7–12
[53] Sriram K. Rajamani, G. Ramalingam, Venkatesh Prasad Ranganath, and Kapil Vaswani, (2009) "Isolator: dynamically ensuring isolation in comcurrent programs" In ASPLOS 09: Architectural Support for Programming Languages and Operating Systems, pp. 181–192.
[54] Shapiro J S, Doerrie M S, Northup E, Sridhar S, Miller M (2004) "Towards a verified, general-purpose operating system kernel" In: G Klein, ed., Proceedings of the NICTA Formal Methods Workshop on Operating Systems Verification, Technical Report 0401005T-1, NICTA, Sydney, Australia
[55] McMillan, Kenneth L. (1993) "Symbolic model checking", Springer US.
[56] Clarke, Edmund M., Orna Grumberg, and Doron Peled, (1999) "Model checking", MIT press.
[57] Necula, George C. (2002) "Proof-carrying code design and implementation", Springer, Netherlands.
[58] Appel, Andrew W., (2001) "Foundational proof-carrying code" Logic in Computer Science, 2001 Proceedings 16th Annual IEEE Symposium on. IEEE.
[59] Holzmann, Gerard J. (2002) "Static source code checking for user-defined properties."Proc. IDPT, Vol. 2.
[60] Leino, K. Rustan M. Dafny, (2010) "An automatic program verifier for functional correctness. Logic for Programming", Artificial Intelligence, and Reasoning, Springer Berlin Heidelberg.
[61] Budd, Timothy A., et al., (1980) "Theoretical and empirical studies on using program mutation to test the functional correctness of programs." Proceedings of the 7th ACM SIGPLAN SIGACT symposium on Principles of programming languages. ACM.
[62] J. S. Shapiro. The EROS Web Site. http://www.eros-os.org. (Link visited March 2015)
[63] Charles Landau The CapROS Web Site. http://www.capros.org (Link visited March 2015)
[64] Shapiro J S Coytos web site, http://www.coyotos.org/ (Link visited March, 2015)
[65] Shapiro, J. S., Northup, E., Doerrie, M. S., Sridhar, S., Walfield, N. H., & Brinkmann, M. Coyotos (2007) "microkernel specification", The EROS Group, LLC, 0.5 edition.
[66] Sharma, Rahul, Aditya V. Nori, and Alex Aiken, (2014) "Bias-variance tradeoffs in program analysis", ACM SIGPLAN Notices, Vol. 49(1). ACM.
[67] Sanán, David, Andrew Butterfield, and Mike Hinchey, (2014) "Separation Kernel Verification: The Xtratum Case Study" Verified Software: Theories, Tools and Experiments. Springer International Publishing, pp. 133-149.
[68] Lange, Matthias, et al., (2011) "L4Android: a generic operating system framework for secure smartphones." Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices. ACM.
[69] Shi, Jianqi, et al., (2012) "ORIENTAIS: Formal verified OSEK/VDX real-time operating system." Engineering of Complex Computer Systems (ICECCS), 17th International Conference on. IEEE.
[70] Qian, Zhenjiang, Hao Huang, and Fangmin Song, (2013) "VTOS: Research on Methodology of "Light-Weight" Formal Design and Verification for Microkernel OS." Information and Communications Security. Springer International Publishing, pp. 17-32.
[71] Woodruff, Jonathan D., (2014) " CHERI: A RISC capability machine for practical memory safety" University of Cambridge, Computer Laboratory, Technical Report, UCAM-CL-TR-858.
[72] Akella, Ravi, and Bruce M. McMillin, (2013) "Modeling and verification of security properties for critical infrastructure protection", Proceedings of the Eighth Annual Cyber Security and Information Intelligence Research Workshop. ACM.

[73] Heiser, Gernot, et al., (2013) "RapiLog: reducing system complexity through verification." Proceedings of the 8th ACM European Conference on Computer Systems, ACM.
[74] Sutisna Nana, et al., (2014) “Live demonstration: Hardware-software co-verification for very large scale SoC using synopsys HAPS platform, Circuits and Systems” (APCCAS), IEEE Asia Pacific Conference, IEEE.
[75] Ströder, Thomas, et al., (2014) “Proving termination and memory safety for programs with pointer arithmetic, Automated Reasoning”, Springer International Publishing, pp.208-223.
[76] Kobashi, Jumpei, Satoshi Yamane, and Atsushi Takeshita, (2014) “Development of SMT-Based Bounded Model Checker for embedded assembly program” Consumer Electronics (GCCE), IEEE 3rd Global Conference.
[77] Lange, Matthias, et al. (2011) "L4Android: a generic operating system framework for secure smartphones." Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices. ACM.
[78] Shrestha Pradhumna Lai, Michael Hempel, and Hamid Sharif, (2014) "Towards a unified model for the analysis of timing-based covert channels."Communications (ICC), 2014 IEEE International Conference, IEEE.
[79] Mitsch Stefan, Sarah M. Loos, and André Platzer, (2012) "Towards formal verification of freeway traffic control." Cyber-Physical Systems (ICCPS), 2012 IEEE/ACM Third International Conference.

Authors

Kushal Anjaria is a PhD scholar at the department of computer science and engineering of Defence Institute of Advanced Technology, Pune-India. He received M Tech in computer science and engineering from Manipal Institute of Technology, Manipal in 2012. His work is currently focused on operating system security and formal verification.

Arun Mishra is an assistant professor at the department of computer science and engineering of Defence Institute of Advanced Technology, Pune-India. He got his PhD in computer science from Motilal Nehru National Institute of Technology, Allahabad (India). His research activity is based on Automated Systems, Trusted Computing, Secure Software Engineering, Formal Modelling and Component based Software Engineering. More information is available at: http://www.diat.ac.in/index.php?option=com_content&view=article&id=206&Itemid=359&Itemid=267