Global Journal of Pure and Applied Mathematics.
ISSN 0973-1768 Volume 12, Number 3 (2016), pp. 2479–2501

http://www.ripublication.com/gjpam.htm

# Correctness of data security tools for protection against unauthorized access and their interaction in GNU/Linux

**Andrey Mikhailovich Kanner**

*National Research Nuclear University MEPhI,*
*Russian Federation, 115409,*
*Moscow, Kashirskoe highway, 31.*

**Abstract**

The article considers applicability of security models to GNU/Linux protection and correctness of the embeddable data security tools for protection against unauthorized access (DST PUA) implementing them, both in terms of formal models, and in terms of interaction between several security tools and access control policies. Existing DST PUA for GNU/Linux-based OS represent a combination of several access control policies, and in addition thereto, other security tools can be embedded in the OS. In such a case correctness of the combination and consistency of all the security policies operating in GNU/Linux are not justified. Moreover, there is no understanding of sufficiency of particular DST PUA to guarantee security of the data processed in GNU/Linux. The article provides ways for eliminating the above deficiencies, considers applicability of existing formal security models and defines a model, which is the most suitable for protecting GNU/Linux and ensuring security of the stored and processed data from unauthorized access. In addition, a mathematical model of one of the DST PUA has been constructed, and its correctness has been justified in accordance with formal security models of computer systems and in combination with other security tools operating in GNU/Linux. The author takes the software and hardware complex "Accord-X" developed by him as an example of a DST PUA. At the same time, the proposed mathematical model and the approach to justification of correctness can be easily adapted to any DST PUA operating in GNU/Linux.

**AMS subject classification:** 08A70, 68M07, 93A30, 93C83.
**Keywords:** Security models, an isolated program environment, embeddable data security tools, correctness of a DST PUA, guarantee of data security in GNU/Linux.

## 1. Introduction

Taking into account the recent tendency to widely use GNU/Linux-based operating systems (implementation of "free" software in educational and public institutions; all-round embedding of GNU/Linux in various portable electronic devices), and according to the latest statistical studies (see Table 1 and Figure 1), more and more common information, confidential and personal data, as well as data representing a state secret are stored and processed in such OS.

Various GNU/Linux distributions provide a wide range of embedded security tools protecting personal data. If such tools are adequately configured, the GNU/Linux-based OS are quite secure. However, such protection can be sufficient only if you do not take into account the possibility of unauthorized access to the computer or to the data carrier (in such a case the operation principle of the security tools can be changed).

Moreover, while creating and implementing data security tools (DST), there arises a problem of correct implementation of the security model used therein. First, security tools should be embedded in a real computer system (CS), so that the formal model corresponding to the used security policy will be a homomorphous mapping of a real CS [4]. Second, formal security models used in various DST should form a protection system possessing the completeness property (in terms of completeness of the mechanisms controlling any possible types of subject-to-object access like in the covered security system model [9]) and guarantee data protection from unauthorized access (UA).

In practice, it is quite difficult to construct a homomorphism of CS to a mathematical model. In particular, this is due to the fact that mathematical models can set strict requirements to determining the manner of operation of the security system, which can make it difficult to implement them in a real CS. Thus, while implementing the CS and security systems, there arises a problem of their adequacy and correspondence to the formal security model.

Moreover, one should also take into account the issue of combining several implemented security models (including while using several DST). In GNU/Linux, default security mechanisms represent a combination of several formal security models (attribute-based discretionary access control, role-based access control using groups, etc.). At the same time, use of, for example, embeddable DST PUA does not cancel, but supplements default security mechanisms of the OS. Thus, if we speak about GNU/Linux, the issue of correctness and consistent interaction of security policies of all the embedded DST PUA is very topical.

Thus, there has appeared a contradiction in the society between the need to use GNU/Linux and protect data therein, on the one hand, and insufficiency of default security tools in GNU/Linux and inability to justify sufficiency and correctness of the used DST PUA to guarantee data protection, on the other hand. The above contradiction can be resolved by constructing a mathematical model of a DST PUA, with a proof of its correctness in accordance with formal security models and in combination with other security tools of GNU/Linux.

The following scientists have made a great contribution in the development of theoretical fundamentals of computer security, including in terms of formal security models

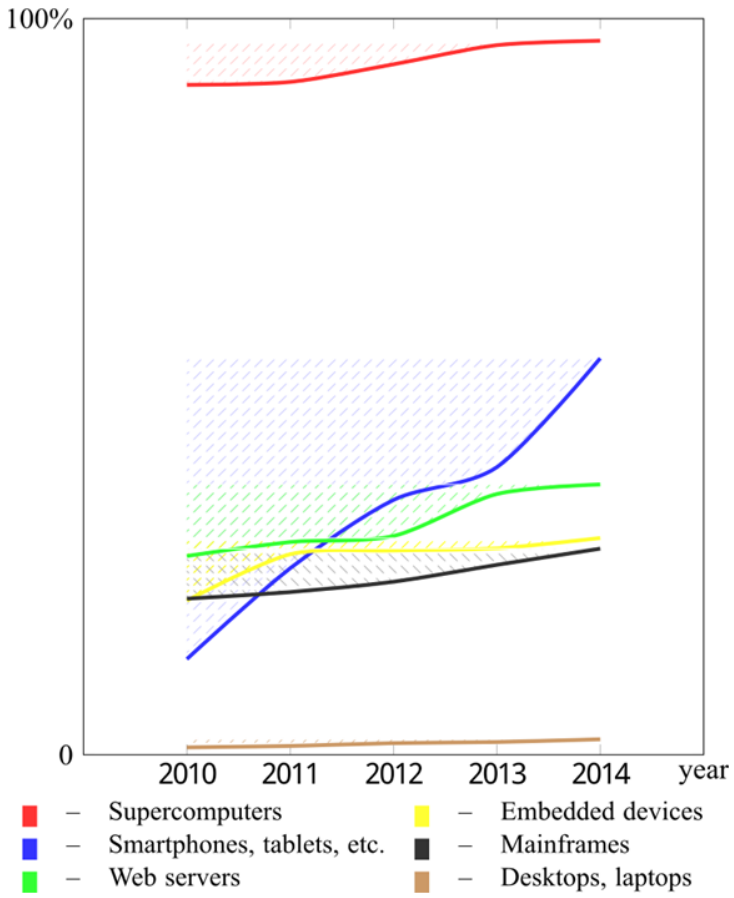


Figure 1: Changes in the rate of using GNU/Linux as an OS for various categories of computers over 2010–2014.

of the CS and their practical use: M. Harrison, W. Ruzzo and J. Ullman [8], D. E. Bell and L. J. LaPadula [1], R. Sandhu, D. F. Ferraiolo and D. R. Kuhn [20], D. E. Denning [3], J. McLean [18], P. N. Devyanin [4], A. Yu. Scherbakov [21, 22], V. A. Konyavsky [15], D. P. Zegzhda [28].

M. Harrison, W. Ruzzo, J. Ullman, D. E. Bell, L. J. LaPadula, R. Sandhu, D. F. Ferraiolo, D. R. Kuhn, D. E. Denning and J. McLean are the founders of classical security model.

A. Yu. Scherbakov's works provide a model guaranteeing inability to change the security policy applied in the CS.

In their works, P. N. Devyanin and D. P. Zegzhda have analyzed classical security models of the CS, and described some results of their practical application.

A great contribution in the development of hardware data security tools and the theory of trusted computing environment was made by V. A. Konyavsky.

However, known works pay little attention to correct combination of formal security models in real CS (including GNU/Linux), as well as to the issue of sufficiency

Table 1: Statistical data on use of GNU/Linux as an OS for various categories of computers (as of the beginning of 2015).

| | **Computer category** | **Data source** | **OS based on Linux kernel** | **OS based on BSD and other Unix** | **Windows** |
|---|---|---|---|---|---|
| **1.** | Supercomputers | TOP500 | 97.0% | 2.4% | 0.2% |
| **2.** | Smartphones, tablets, etc. | StatCounter Global Stats | 53.86% | 31.10% | 1.87% |
| **3.** | Web servers | W3Techs | 36.72% | 30.18% | 33.10% |
| **4.** | Embedded devices | UBM Electronics | 29.44% | 4.29% | 11.65% |
| **5.** | Mainframes | Gartner | 28% | 72% | – |
| **6.** | Desktops, laptops | Net Applications | 1.34% | 7.21% | 91.45% |

of particular models (and tools implementing them) to guarantee data protection from unauthorized access.

The foresaid leads to the following scientific and technical tasks: to determine the most suitable security model guaranteeing data security in GNU/Linux; to construct a mathematical model of a DST PUA embedded in a real CS; to justify correct operation of the DST PUA in accordance with formal security models and in combination with other security tools of the CS.

## 2. Method

### 2.1. Applicability of CS formal security models to GNU/Linux

In the modern theory of computer security, for the purposes of formal modeling, a CS is most often represented in the form of an abstract system, each state of which is fully described by subject-to-object access [26]. At the level of axioms, the assumption is accepted that all the security issues in the CS are described by subject-to-object access, and all the actions can be initiated only by subjects by accessing the CS entities.

Access of any subjects to objects is allowed by the access control subsystem only if the subjects possess the required authorities. The allowed access rights are described in the CS security policy [11], which represents a combination of rules regulating management and protection of the CS resources [4]. Compliance with such rules ensures protection from threats influencing the main security properties of information (confidentiality, integrity and availability). A formal expression of the security policy is called a security policy model.

The main purpose of describing the CS security policy as a formal model is to develop and prove the security criterion and to formally prove correspondence of the CS with this criterion, subject to certain conditions and limitations [4]. In practice, meeting the security criterion means that at any time given, only particular subjects of the CS can access objects, with only authorized access allowed.

The following are the main types of access control policies (their security models are called classical) determining the manner of setting the allowed access rights: discretionary access control [8], mandatory access control [1], role-based access control [20].

Alongside with the above access control policies, there are the following security models: classical ones and the DP-security models of information flows control [2, 4, 7, 18]; models of isolated program environment (the subject-oriented model or the SO-model [5, 21, 22] and its extension [14, 15]).

The models of access control policies and other security models are considered in detail in a number of works [4, 27], and their detailed description is beyond the scope of this article. Below, there are results of the analysis carried out to determine applicability of these models to protecting GNU/Linux.

For some security models, it is difficult to analyze how access control rules influence security of the CS (for example, for most of the discretionary access control models); for some models, the security check is an algorithmically undecidable problem in terms of the algorithm theory [16]; for other models, there are quite effective check algorithms.

At the same time, for example, the Bell-LaPadula model determines what properties the CS states and its entities' actions should have, but does not set an accurate order of the security system's actions performed in response to access requests. Due to the possibility to temporarily reduce the subject's access level (i.e. clearance), the security of the CS can be guaranteed only in case of a "lack of a memory"(i.e. in case of inability to write data in the object of a lower confidentiality level in any of the next states of the CS)In information flow security models, in order to justify security of the CS, subjects and objects should be ensured informational nondeducibility or informational noninterference, which is difficult to accomplish in modern CS.

In all of the above security models, there should be at least the following entities of the CS:

- An active subject controlling all the activities performed by subjects in relation to objects;
- An object containing information about the allowed and the prohibited activities of subjects in relation to objects.

Some models (for example, the SO-model) set quite abstract requirements for such entities, compliance with which guarantees inability to change the security policy applied in the CS without authorization. Other models can set requirements for interaction with such entities (the so-called policies of "secure administration" and "absolute division of administrative and user authorities"), but some security models do not pay attention to administration issues at all.

One of the fundamental conditions in classical CS access control policies is fixation of subjects' and objects' sets. This is practically impossible to meet this requirement in real CS – while the system is transiting to new states, both new objects and subjects can appear.

A number of other models consider the processes of generating a subject by another subject, or generating a subject out of an object (as a result of execution access). Here a number of questions arises – "what generates the first subject?", "when does activation of subjects start?" Many models do not give an answer to these questions, and there appears a problem like "the chicken or the egg". There is no understanding of the reasons for subject emergence: is this another subject or an activated initial object? The dilemma of subjects' and objects' priority is solved in detail only in the SO-model of the isolated program environment (IPE) and its extension – the trusted computing environment (TCE).

While considering the classical Bell-LaPadula model and the Biba integrity model, the problem of combining security tools protecting from threats to confidentiality and integrity becomes apparent [19]. If both threats are to be averted simultaneously (which is required in real CS), use of the rules prescribed by the Bell-LaPadula and Biba models can lead to a situation, when the security and the integrity levels are used in opposite ways [19, 23].

Thus, the following issues, including those characteristic of GNU/Linux protection, remain open for all the CS security models:

1. Correctness of the access control subsystem being implemented (where and how should security policy rules be stored, the question of secure administration);

2. Determination of the required level of embedding security tools and controlling their correct operation (including departure from the notion of an abstract secure initial state);

3. Correctness of the security policy in case of a change in subjects' or objects' sets;

4. Necessity and sufficiency of access attributes, completeness of controlled access types (except for the considered basic ones – read, write/append and exec);

5. Simultaneous application of mechanisms ensuring confidentiality and integrity of objects.

It is impossible to unambiguously answer the question, what formal security model is better in general. As to the above security models, discretionary and mandatory access control (due to their wide use in various systems), the SO-model and its extension up to the TCE, considering not only the subject-to-object access but also the environment where it takes place, as well as the DP-models (which are mainly designed not for local use, but ensure protection from all types of information flows) are of particular interest. Moreover, the SO-model describes rules of interaction between CS subjects, ensuring inability to influence the security system or modify its parameters. However, the choice of one or another security model depends mainly on one's preferences, as well as on implementation of the CS being protected.

The SO-model of IPE and its extension up to the TCE, in combination with one or more access control policies (the SO-model is invariant in relation to the access control

policies used) can be considered as the most suitable to protect GNU/Linux-running computers. If all the above questions, which have remained open for security models, are solved, using this combination it will be possible to ensure the required and sufficient security level both for the information and the user environment, in which the information is processed in GNU/Linux.

## 2.2. Correctness of embeddable DST's operation in GNU/Linux

### 2.2.1 Formal description of the SO-model of IPE

Let us consider the software and hardware complex "Accord-X" developed by the author [12, 13] as an example of a DST PUA. To justify correctness of implementation of its security mechanisms, let us construct a homomorphism with the formal SO-model of IPE (as the most suitable for protection of GNU/Linux). To do this, let us at first formally describe this model in accordance with [4, 5, 21, 22].

**Definition 2.1.** Let $S_{so} = S^l_{so} \bigcup S^n_{so} \bigcup \{s^{ssm}_{so}, s^{osm}_{so}\}$ denote a set of subjects within the SO-model of IPE consisting of:

- A set of allowed ($S^l_{so}$) and prohibited ($S^n_{so}$) subjects;
- Subjects' security monitor ($s^{ssm}_{so}$) and objects' security monitor ($s^{osm}_{so}$).

**Definition 2.2.** Let $O_{so} = O^a_{so_t} \bigcup O^{na}_{so_t} \bigcup \{[s^{ssm}_{so}]_t, [s^{osm}_{so}]_t\}$ denote a set of objects within the SO-model of IPE consisting of:

- A set of objects associated (functionally associated or associated as data objects [4]) with some subjects at a time point $t \geq 0$ ($O^a_{so_t}$);
- A set of objects not associated with subjects at a time point $t \geq 0$ ($O^{na}_{so_t}$);
- Objects associated with subjects' ($[s^{ssm}_{so}]_t$) and objects' ($[s^{osm}_{so}]_t$) security monitors at a time point $t \geq 0$.

**Definition 2.3.** Let $\emptyset_{so}$ denote the "zero" element within the SO-model of IPE. This element represents a pseudo entity having no authorities in the system, containing "empty" data and not being subject to any modifications (Note 1).

In this case all the system's entities form a set $S_{so} \bigcup O_{so} \bigcup \{\emptyset_{so}\}$.

**Definition 2.4.** Let us introduce the operation $Create_{so}(s, o) \rightarrow s'$ creating the subject $s'$ out of the object $o$ under the activating influence of the subject $s$, where $s, s' \in S_{so}$, $o \in O_{so}$ (if $s' \neq \emptyset_{so}$ creation was successful; if $s' = \emptyset_{so}$, creation is impossible).

In this case at the moment of creating the subject $s'$ (for example, $t' \geq 0$), while executing $Create_{so}(s, o) \rightarrow s'$ the source object is associated with $s'$ at the time point $t'$, i.e. $o \in [s']_{t'} \subseteq O^a_{so_{t'}}$.

**Definition 2.5.** Let us introduce the operation $Stream_{so}(s, o) \to o'$ determining emergence of an information flow from the object $o$ to the object $o'$ under the influence of the subject $s$, which results in transformation of information in the object $o'$, where $s \in S_{so}$ and $o, o' \in O_{so}$ (in this case $o$ and $o'$ can be both associated and not associated with $s$).

**Definition 2.6.** Let $P_{so}$ denote a set of information flows for fixed decomposition of the system into subjects and objects at any time point, with both prohibited ($N_{so}$) and allowed ($L_{so}$) information flows, in which case $P_{so} = N_{so} \bigcup L_{so}$ and $N_{so} \bigcap L_{so} = \emptyset$.

**Definition 2.7.** To filter any emerging information flow between the CS objects, the objects' security monitor (OSM) executes the function $Filter\_access_{so}()$, such that:

$$Filter\_access_{so}(s, o, o') = \begin{cases} Stream_{so}(s, o) \to o', & \text{if } Stream_{so}(s, o) \to o' \in L_{so} \\ \emptyset_{so}, & \text{if } Stream_{so}(s, o) \to o' \in N_{so} \end{cases}$$

The information about the allowed values of the mapping $Stream_{so}()$ is contained in the control object associated with the OSM – $o_{so}^{osm} \in [s_{so}^{osm}]_t,\ \forall t \geq 0$.

In case of a change in the objects associated with the OSM, its properties can change (i.e. there can appear information flows from $N_{so}$). In this connection, the SO-model of IPE introduces the following definitions and proposition.

**Definition 2.8.** The subjects $s, s' \in S_{so}$ are correct with respect to each other, when at any time point there is no information flow in the CS between any $o, o' \in O_{so}$, associated with $s$ and $s'$, correspondingly, i.e. the following condition is met: $\forall t \geq 0$ and $\forall o \in [s]_t,\ o' \in [s']_t$ there is no s" : there exist $Stream_{so}(s", o')$ or $Stream_{so}(s", o)$.

The subjects $s, s' \in S_{so}$ are called absolutely correct, if they are correct and their associated objects' sets are disjoint, i.e. $\forall t \geq 0 :\ [s]_t \bigcap [s']_t = \emptyset$.

**Definition 2.9.** To control creation of subjects in the CS, the subjects' security monitor (SSM) executes the function $Filter\_spawn_{so}()$, such that:

$$Filter\_spawn_{so}(s, o, s') = \begin{cases} Create_{so}(s, o) \to s', & \text{if } s' \in S_{so}^{l} \\ \emptyset_{so}, & \text{if } s' \in S_{so}^{n} \end{cases}$$

The information about the allowed values of the mapping $Create_{so}()$ is contained in the control object associated with the SSM – $o_{so}^{ssm} \in [s_{so}^{ssm}]_t,\ \forall t \geq 0$.

**Definition 2.10.** The CS is called closed in terms of subject creation (possesses closed program environment), when the SSM operates therein, and the function $Filter\_spawn_{so}()$ from Definition 2.9 is implemented (i.e. the SSM selects the subset $S_{so}^{l}$).

**Definition 2.11.** The program environment is called isolated (absolutely isolated), when it is closed in terms of subject creation, and the subjects being created are correct (absolutely correct) with respect to each other, the SSM and the OSM.

**Lemma 2.12.** The OSM in the absolute IPE allows creating flows only from the set $L_{so}$.

The proof of the lemma is provided in [4] and is based on the fact that the objects associated with the OSM can be changed only by the OSM (i.e., it is impossible to change the state of the OSM without authorization).

To describe requirements to the maintenance of IPE in the CS, the SO-model provides a definition of subject creation with control over invariability of their source objects, and a Basic Theorem of IPE.

**Definition 2.13.** Creation of the subject $s' \in S_{so}$ with the help of the operation $Create_{so}(s, o) \rightarrow s'$ is called creation with control over invariability of the source object, when for any given time $t > t_0$ ($t_0$ – the time when the operation is executed) creation of the subject $s'$ is possible only if the object $o$ is identical at the time points $t_0$ and $t$.

**Theorem 2.14. (Basic Theorem of IPE).** If beginning from the time point $t_0$, subjects are created only with control over invariability of the source object in the absolute IPE, and all the created subjects are absolutely correct with respect to each other and the existing subjects, then the program environment will remain the absolute IPE at any time point $t > t_0$.

The proof of this Basic Theorem of IPE is provided in [4] and it is based on the fact that if the conditions of Lemma 2.12 are met, there can exist flows in the CS changing the state of only those objects that are not associated with subjects (i.e. the set of IPE subjects is not extendable, and the program environment always remains absolutely isolated).

If the conditions of Theorem 2.14 are met, "closure" of the absolute IPE is maintained in the CS, which guarantees implementation of the security policy set in the OSM. In fact, the conditions required to maintain the IPE "closed", impose some limitations on the functions $Filter\_access_{so}()$ and $Filter\_spawn_{so}()$. Let us describe them formally below.

Let $t_{sandbox}$ denote a time point, when the absolute IPE is achieved in the CS, and let $o[t]$ denote a state of the object $o \in O_{so}$ at the time point $t \geq 0$.

**Definition 2.15.** Let us define the function $Filter\_spawn_{so}^{sandbox}()$ limiting the standard function $Filter\_spawn_{so}()$ with a requirement to create subjects, with control over invariability of their source objects, i.e.:

$$\forall t > t_{sandbox} : \; Filter\_spawn_{so}^{sandbox}(s, o, s') = \begin{cases} Create_{so}(s, o) \rightarrow s', & \text{if } s' \in S_{so}^{l} \\ & \text{and} \\ & o[t] = o[t_{sandbox}] \\ \emptyset_{so}, & \text{otherwise} \end{cases}$$

**Definition 2.16.** Let us define the function $Filter\_access_{so}^{sandbox}()$ limiting the standard function $Filter\_access_{so}()$ with a requirement to prohibit emergence of an information flow between the objects associated with any subjects (in fact, with a requirement of

absolute correctness, since the system is absolutely isolated at the time $t_{sandbox}$):

$$\forall t > t_{sandbox} : \ Filter\_access_{so}^{sandbox}(s, o, o') = \begin{cases} Stream_{so}(s, o) \to o', & \text{if } Stream_{so}(s, o) \to o' \in L_{so} \\ & \text{and at the same time} \\ & o \notin [s']_t, o' \notin [s"]_t, \ \forall s', s" \in S_{so} \\ \emptyset_{so}, & \text{otherwise} \end{cases}$$

**Proposition 2.17.** If the OSM and the SSM implement the functions $Filter\_access_{so}^{sandbox}()$ and $Filter\_spawn_{so}^{sandbox}()$ correspondingly in the absolute IPE, beginning from the time point $t_{sandbox}$, then at any given time $t > t_{sandbox}$, the program environment will also remain the absolute IPE.

The correctness of this proposition follows directly from the conditions of Theorem 2.14 and Definitions 2.15 and 2.16

**Corollary 2.18.** If the OSM and the SSM implement the functions $Filter\_access_{so}^{sandbox}()$ and $Filter\_spawn_{so}^{sandbox}()$ correspondingly in the absolute IPE, beginning from the time point $t_{sandbox}$, only allowed information flows from the set $L_{so}$ will appear in the CS.

The correctness of Corollary 2.18 follows directly from Proposition 2.17 and Lemma 2.12.

To achieve IPE (absolute IPE) in the SO-model, it is necessary to duly implement the OSM and the SSM and ensure pairwise correctness (absolute correctness) of subjects with respect to each other, the OSM and the SSM. IPE implementation can consist of two stages: predetermined execution of the initial stage with activation of the OSM and the SSM, and work in the IPE mode with control over invariability of source objects.

### 2.2.2 Correctness of the DST PUA in GNU/Linux in accordance with the SO-model of IPE

Let us now justify correctness of the security mechanisms implemented in the complex "Accord-X" developed by the author [12, 13] and consisting of a hardware component (the trusted start-up hardware module (TSHM)), and a software component (an access control subsystem in GNU/Linux). The justification will be provided in the form of a proposition, in which a mathematical model of Accord-X will be constructed, which, in general, can be easily adapted to any DST PUA in GNU/Linux.

**Proposition 2.19.** There is a homomorphism of the following algebraic systems:

– The SO-model of the absolute IPE:

$$CS_{so} = \left\langle \begin{array}{l} S_{so} \bigcup O_{so} \bigcup \{\emptyset_{so}\}, \\ \{Create_{so}(), \ Stream_{so}(), \ Filter\_spawn_{so}^{sandbox}(), \ Filter\_access_{so}^{sandbox}()\} \end{array} \right\rangle$$

– The CS (GNU/Linux) with the embedded complex "Accord-X":

$$CS_{acx} = \left\langle \begin{array}{l} S_{acx} \bigcup O_{acx} \bigcup \{\emptyset_{acx}\}, \\ \{Create_{acx}(),\ Stream_{acx}(),\ Filter\_spawn_{acx}(),\ Filter\_access_{acx}()\} \end{array} \right\rangle$$

In this case, the following conditions are met for the homomorphism $\varphi : CS_{so} \to CS_{acx}$:

1. $\varphi(Create_{so}(s_{so}, o_{so}) \to s'_{so}) = Create_{acx}(\varphi(s_{so}), \varphi(o_{so})) \to \varphi(s'_{so})$ for any $s_{so}, s'_{so} \in S_{so}$ and $o_{so} \in O_{so}$;

2. $\varphi(Stream_{so}(s_{so}, o_{so}) \to o'_{so}) = Stream_{acx}(\varphi(s_{so}), \varphi(o_{so})) \to \varphi(o'_{so})$ for any $s_{so} \in S_{so}$ and $o_{so}, o'_{so} \in O_{so}$;

3. $\varphi(Filter\_spawn_{so}^{sandbox}(s_{so}, o_{so}, s'_{so})) = Filter\_spawn_{acx}(\varphi(s_{so}), \varphi(o_{so}), \varphi(s'_{so}))$ for any $s_{so}, s'_{so} \in S_{so}$ and $o_{so} \in O_{so}$;

4. $\varphi(Filter\_access_{so}^{sandbox}(s_{so}, o_{so}, o'_{so})) = Filter\_access_{acx}(\varphi(s_{so}), \varphi(o_{so}), \varphi(o'_{so}))$ for any $s_{so} \in S_{so}$ and $o_{so}, o'_{so} \in O_{so}$.

This means that $CS_{acx}$ is built in a way similar to $CS_{so}$ in terms of the operations $Filter\_spawn_{acx}()$, $Filter\_access_{acx}()$ and corresponds to the Basic Theorem of IPE.

*Proof.* Further the following designations will be used within $CS_{acx}$, identically to appropriate designations within $CS_{so}$: $S_{acx}$, $O_{acx}$, $\emptyset_{acx}$, $Stream_{acx}(s, o) \to o' \in P_{acx} = N_{acx} \bigcup L_{acx}$, $Create_{acx}(s, o) \to s'$, $Filter\_access_{acx}(s, o, o')$, $Filter\_spawn_{acx}(s, o, s')$.

In accordance with the introduced designations, as well as Definitions 2.4, 2.5, 2.15 and 2.16, to construct the homomorphism $\varphi : CS_{so} \to CS_{acx}$, it will be necessary and sufficient to show correspondence between the operations $Create_{so}()$, $Stream_{so}()$, $Filter\_spawn_{so}^{sandbox}()$, $Filter\_access_{so}^{sandbox}()$ and $Create_{acx}()$, $Stream_{acx}()$, $Filter\_spawn_{acx}()$, $Filter\_access_{acx}()$. To do this, let us at first determine composition of subjects and objects of $CS_{acx}$ more correctly.

Objects of the OS file system associated ($O^{a}_{acx_t}$) or not associated ($O^{na}_{acx_t}$) with any subjects at the time point $t \geq 0$ initially act as objects in GNU/Linux and the access control subsystem of Accord-X.

Users of the OS (including the so-called "pseudo users" – daemons and services) and processes (including system processes) running with user privileges, initially act as actual subjects in GNU/Linux and the access control subsystem of Accord-X. However, as to emergence of information flows (which flow between objects, with subjects having no "memory"), it will be more right to consider only the following (sometimes abstract) entities as subjects:

– Real users of GNU/Linux interact with the OS;

– System daemons and services operating in GNU/Linux in accordance with a predetermined operation algorithm (not interactively).

In this case, the processes run on behalf of the above subjects act as objects functionally associated with them. Further we will assume that existence of one or another subject in $CS_{acx}$ at the time point $t \geq 0$ is characterized by availability of process objects associated therewith at the time point $t$ (creation of a subject – emergence of the first process associated therewith).

In addition, in GNU/Linux and the access control subsystem of Accord-X, there is a subject playing the role of the OSM and the SSM ($s_{acx}$– a security kernel implementing security policies and controlling creation of subjects [12, 13]), as well as objects associated with it at the time point $t \geq 0$ ($[s_{acx}]_t$ – processes of administration utilities, configuration files with access rights and rules of subject creation).

In accordance with the accepted description of subjects and objects, their sets in $CS_{acx}$ can be described as follows:

1. $S_{acx} = S^l_{acx} \bigcup S^n_{acx} \bigcup \{s_{acx}\}$, where:
   - $S^l_{acx} = S^l_{users} \bigcup S^l_{shadows}$ is the set of subjects allowed in $CS_{acx}$, $S^l_{users}$ is the set of allowed users, $S^l_{shadows}$ is the set of allowed system services and daemons;
   - $S^n_{acx}$ is the set of subjects prohibited in $CS_{acx}$;
   - $s_{acx}$ is the security kernel (the OSM and the SSM).
2. $O_{acx} = O^a_{acx_t} \bigcup O^{na}_{acx_t} \bigcup \{[s_{acx}]_t\}$ for $t \geq 0$, where:
   - $O^a_{acx_t} = O^a_{func_t} \bigcup O^a_{data_t}$ is the set of objects associated with some subjects at the time point $t$, $O^a_{func_t}$ is the set of functionally associated objects, $O^a_{data_t}$ is the set of objects associated as data;
   - $O^{na}_{acx_t}$ is the set of objects not associated with subjects at the time point $t$;
   - $[s_{acx}]_t$ is objects associated with $s_{acx}$ at the time point $t$.

   In this case all the entities of $CS_{acx}$ form the set $S_{acx} \bigcup O_{acx} \bigcup \{\emptyset_{acx}\}$.

Any subjects' requests for access to objects – read, write, execute, create/delete objects, etc. – act as the operation $Stream_{acx}(s, o) \rightarrow o'$ in GNU/Linux. Let us formally describe any possible types of access in terms of emergence of information flows $Stream_{acx}(s, o) \rightarrow o'$ for the subject $s \in S_{acx}$ and the time of execution $t \geq 0$:

1. $read()$: the subject $s$ reads data from the object $o$ and writes them in $o'$ ($o' \in [s]_t$, if $s \neq s_{acx} : \ o' \in O^a_{func_t}$; $o \in O_{acx}$, and can influence $o'$ – i.e. probably $o \in [s]_t$, if $s \neq s_{acx} : \ o \in O^a_{data_t}$);
2. $write()$: the subject $s$ writes data from $o$ in $o'$ ($o \in [s]_t$, if $s \neq s_{acx} : \ o \in O^a_{func_t}$; $o' \in O_{acx}$);

3. $create()$: similar to the write operation, but $o'$ is added in $O_{acx}$ ($o \in [s]_t$, if $s \neq s_{acx} : o \in O^a_{func_t}$);

4. $delete()$: similar to the write operation, but $o'$ is deleted from $O_{acx}$ ($o \in [s]_t$, if $s \neq s_{acx} : o \in O^a_{func_t}$);

5. $exec()$: the subject $s$ loads a binary image for execution from $o$ to $o'$ ($o \in [s]_t$ if $s \neq s_{acx} : o \in O^a_{data_t}$ at the time of execution; $o' \in [s]_t$, if $s \neq s_{acx} : o' \in O^a_{func_t}$ including before and after $t$);

6. $fork()$: creation of $o'$ and adding it to $[s]_{t+1}$ ($o \in [s]_t$, if $s \neq s_{acx} : o \in O^a_{func_t}$; $o' \in [s]_{t+1}$, if $s \neq s_{acx} : o' \in O^a_{func_{t+1}}$).

Other access types can be represented as information flows derived from the information flows listed above.

A combination of several system calls, as well as operations of the OS and Accord-X - $fork()$, $exec()$, $login()$ (identification/authentication) and $setuid()$ (change of the user's ID) is used as the operation $Create_{acx}(s, o) \to s'$ in GNU/Linux and the access control subsystem of Accord-X. In this case $Create_{acx}()$ represents as follows for various subjects [12]:

1. If $s' \in S^l_{users} \subseteq S^l_{acx}$, then $Create_{acx}(s, \{o, o_{id}, o_{auth}\}) \to s'$, where $o \in [s]_t$, if $s \neq s_{acx} : o \in O^a_{func_t}$ is the source object; $o_{id}$ and $o_{auth}$ are the user's identifier and authenticator (provided by the user), with the following operations executed:

   – $login()$consisting of the following stages:

     – User identification: check of the value $o_{id}$ (in case of an error – $Create_{acx}(s, \{o, o_{id}, o_{auth}\}) \to \emptyset_{acx}$);
     – User authentication: check of the value $o_{auth}$ depending on the value of $o_{id}$ (in case of an error – $Create_{acx}(s, \{o, o_{id}, o_{auth}\}) \to \emptyset_{acx}$).

   – One or more operations $fork()$ and $exec()$;

   – The operation of direct creation $setuid()$, in which case $o \in [s]_t$ and $o \in [s']_{t+1}$ for $t \geq 0$, which is the time of execution.

2. If $s' \in S^l_{shadows} \subseteq S^l_{acx}$ and $s$ is authorized to create subjects from $S^l_{shadows}$, then $Create_{acx}(s, o) \to s'$, where $o \in [s]_t$, if $s \neq s_{acx} : o \in O^a_{func_t}$ is the source object, with the following operations executed:

   – One or more operations $fork()$ and $exec()$;

   – The operation $setuid()$ with the properties described above.

3. If $s' \in S^n_{acx}$, then $Create_{acx}(s, o) \to \emptyset_{acx}$.

User subjects can be created in GNU/Linux out of several data objects – */bin/login*, */bin/su*, */usr/bin/sudo*, etc. As to system processes, they can emerge in GNU/Linux without any identification/authentication. In both cases, however, while creating new subjects, the activated objects (associated with creating subjects) involve other objects (PAM-modules from /lib/security/ or /lib64/security/, configuration files in /etc/pam.d/, libraries), which become associated with the creating subject for a certain time period. Let us introduce a designation for such objects: the object $o' \in O_{acx}$ is connected with the object $o \in O^a_{func_t}$ (or $o \in [s_{acx}]_t$) at the time point $t \geq 0$ (hereinafter referred to as $o' \in [o]_t$), if $\exists\, t' < t : \exists\, Stream_{acx}(s, o') \rightarrow o$, where $s \in S_{acx}$ and $o' \in O^a_{data_{t'}}$ (or $o' \in [s_{acx}]_{t'}$ if $s = s_{acx}$).

In addition, in GNU/Linux there is a possibility to create multiple sessions of one user (let us denote existence of a session as $s \in S^{session}_{acx}$, where $S^{session}_{acx} \subseteq S^{l}_{acx}$, with $s$ being added in $S^{session}_{acx}$ in case of success and deleted upon user logout). In general, user sessions are not isolated (due to one session influencing another through common configuration files), and therefore, the possibility of multiple sessions should be limited in GNU/Linux with the help of Accord-X.

To meet the requirement of control over invariability of source objects from Definition 2.15, while creating a new subject, Accord-X should apply dynamic control over integrity of all the object files, out of which subjects can be created (in the process of preceding operations $exec()$), as well as the shared libraries and configuration files used in this process. Let us denote the integrity check for an object as $Check\_integrity(o)$ (and for an object with all other objects connected therewith – $Check\_integrity([o]_t)$). This function takes on the value $True$ only when the object at a current time point is identical to the object at the time point when it is added in the integrity check list. Let us denote presence of the object with all other objects connected there with in the integrity check list as $[o]_t \in Integrity\_list(O_{acx})$.

Following from the definition of $Stream_{acx}()$ and $Create_{acx}()$, as well as Definition 2.16, it becomes obvious that the set of objects associated with a certain subject extends only in case of $fork()$, $exec()$ preceded by $read()$, and $setuid()$. In this case, absolute correctness of all the subjects relative to the objects from $O^a_{func_t}$ (or functionally associated objects from $[s_{acx}]_t$) is guaranteed by the use in GNU/Linux of independent virtual address spaces for any processes in the OS (as well as by the fact that objects can be associated with only one subject at the time point $t \geq 0$).

In this connection, it will be reasonable to consider absolute correctness only with respect to the objects from $O^a_{data_t}$ (or data objects from $[s_{acx}]_t$), which participate in read and execute flows, because in this case, there is a possibility to implement the timing information flow, as a result of which the state of functionally associated objects from $O^a_{func_t}$ or $[s_{acx}]_t$ will be changed. To exclude the possibility of such information flows, it will be reasonable to control access to common objects, allowing their creation and change by some authorized subjects, and to introduce an integrity control mechanism.

To crown it all, in order to meet the requirements of subject creation with control over invariability, and their absolute correctness with respect to each other (conditions of Theorem 2.14), it is necessary to implement the above functions $Create_{acx}()$ and

$Stream_{acx}()$ in Accord-X, limited as follows:

1. Subjects from $S^l_{shadows} \subseteq S^l_{acx}$ can be created only by the subject *shadow:root*, $s_{root} \in S^l_{shadows}$ (the system subject created in GNU/Linux upon initialization or booting the OS kernel, for which the process *init* with $PID = 0$ is one of the associated objects);

2. The operation $login()$ should be implemented while creating subjects from $S^l_{users} \subseteq S^l_{acx}$;

3. A subject from the set $S^l_{acx}$ can be created only if there is no another its session (i.e. the possibility of multiple sessions should be limited);

4. The rights to create and change objects available for reading and executing to all the subjects, as well as objects connected with all the source processes at the time of subject creation, should be provided to a selected subject – a system administrator of GNU/Linux ($s_{admin} \in S^l_{users} \subseteq S^l_{acx}$);

5. The rights to create and change objects associated with the security kernel $s_{acx}$ should be provided to a selected subject – a security administrator ($s_{acx-admin} \in S^l_{users} \subseteq S^l_{acx}$, which can be the same as $s_{admin}$);

6. Integrity of the objects listed in paragraphs 4–5 should be fixed and checked in case of any access thereto. Any authorized changes in these objects should be introduced if there are no other user subjects, and should be followed by an update in all the control sums;

7. Rights of accessing personal objects of user subjects, including configuration files (~/.config, etc.), should be provided only to such subjects.

Thus, formalizing the foresaid, to implement the absolute IPE in $CS_{acx}$, the operations $Filter\_spawn_{acx}()$ and $Filter\_access_{acx}()$ should be implemented as follows:

– $Filter\_spawn_{acx}(s, o, s') = Create_{acx}(s, o) \rightarrow s'$, if $s' \in S^l_{acx}$, $s' \notin S^{session}_{acx}$, and:

  1. If $s' \in S^l_{shadows}$, then $s = s_{root}$;
  2. If $s' \in S^l_{users}$, then $o \sim \{o, o_{id}, o_{auth}\}$ and $login()$ is implemented;
  3. $[o]_t \in Integrity\_list(O_{acx})$ and $Check\_integrity([o]_t) = True$ for $t \geq 0$ – the time when the subject $s'$ is created.

– Otherwise $Filter\_spawn_{acx}(s, o, s') = \emptyset_{acx}$;

– $Filter\_access_{acx}(s, o, o') = Stream_{acx}(s, o) \rightarrow o'$, if $Stream_{acx}(s, o) \rightarrow o' \in L_{acx}$, and:

1. If $Stream_{acx}(s, o) \rightarrow o'$ emerges at the time point $t \geq 0$ as a result of the access $\{read(), exec()\}$ and $\exists\, t' < t,\ s' \in S^{l}_{acx} :\ o \in [s']_{t'}$, then $[o]_t \in Integrity\_list(O_{acx})$ and $Check\_integrity([o]_t) = True$;
2. If $Stream_{acx}(s, o) \rightarrow o'$ emerges as a result of the access $\{read(), exec()\}$ and there is no $s' \in S^{l}_{acx} :\ o \in [s']_{t'}\ \forall t' \geq 0$, then the flow is allowed;
3. If $Stream_{acx}(s, o) \rightarrow o'$ emerges at the time point t $\geq 0$ as a result of the access $\{write(), create(), delete()\}$ and $\exists\, t' < t,\ s' \in S^{l}_{acx} :\ o' \in [s']_{t'}$, then $s = s_{admin}$, $S^{session}_{acx} = \{s_{admin}\}$, $[o']_t \in Integrity\_list(O_{acx})$, $Check\_integrity([o']_t) = True$ and the control sums of $o'$ are updated in $Integrity\_list(O_{acx})$;
4. If $Stream_{acx}(s, o) \rightarrow o'$ emerges as a result of the access $\{write(), create(), delete()\}$ and there is no $s' \in S^{l}_{acx} :\ o' \in [s']_{t'}\ \forall t' \geq 0$, then the flow is allowed;
5. If $Stream_{acx}(s, o) \rightarrow o'$ emerges at the time point t $\geq 0$ as a result of the access $\{write(), create(), delete()\}$ and $\exists\, t' < t :\ o' \in [s_{acx}]_{t'}$, then $s = s_{acx-admin}$, $S^{session}_{acx} = \{s_{acx-admin}\}$, $[o']_t \in Integrity\_list(O_{acx})$, $Check\_integrity([o']_t) = True$ and the control sums of $o'$ are updated in $Integrity\_list(O_{acx})$;

– Otherwise $Filter\_access_{acx}(s, o, o') = \emptyset_{acx}$.

## 3. Results and Discussion

### 3.1. Implementation of the SO-model of IPE in GNU/Linux in practice: Accord-X

The limitations on the functions $Filter\_access_{acx}()$ and $Filter\_spawn_{acx}()$ described in Proposition 2.19 are implemented in Accord-X by correspondingly setting its functions. Thus, $CS_{acx}$ represents a homomorphism of the SO-model of IPE; isolation of subjects' user environments can be shown as a graph (see Figure 2).

In this case $CS_{acx}$ is an extension of the SO-model of IPE up to TCE with additional implementation of the discretionary and the mandatory access control policies, clearance of the core memory in case it becomes free or redistributed, clearance of the residual information in case of deleting objects, as well as mechanisms of dynamic and static integrity control.

As opposed to the SO-model of IPE, in Accord-X, there is a possibility to controllably boot the security kernel and control the integrity of all the objects connected thereto before $CS_{acx}$ is started. For this purpose, trusted start-up of the OS using the TSHM is applied, which method is considered in [13], due to which $CS_{acx}$ is wider than the SO-model (i.e. it allows to arrange TCE). Thus, the "step-by-step" start-up of $CS_{acx}$ comprises the following stages (see Figure 3):

1. Start-up of $CS_{acx}$ begins ($t = 0$);

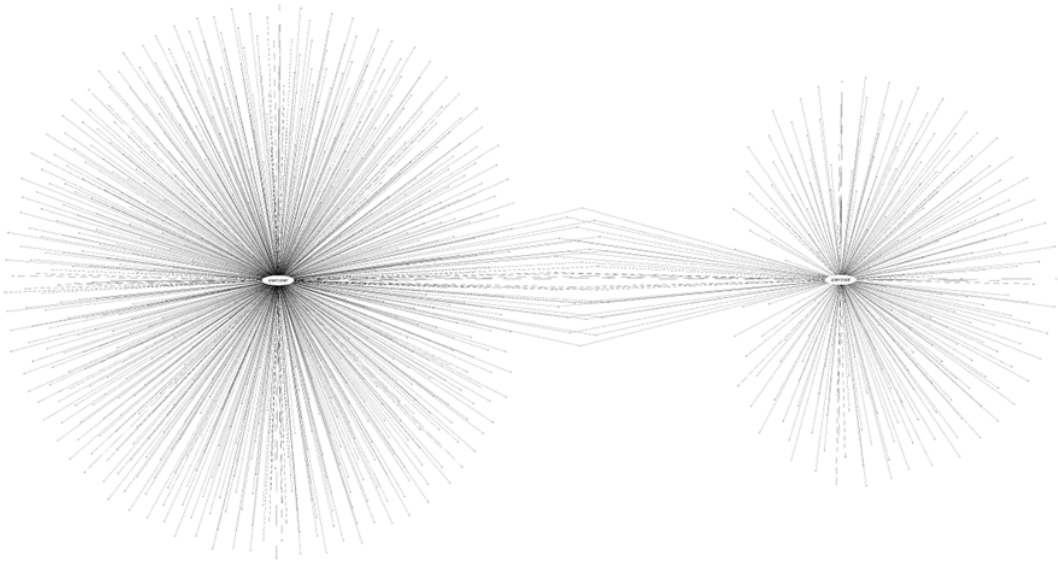

Figure 2: Graph showing access of two subjects to objects in real GNU/Linux (points of the intersections are generally accessible objects, whose integrity is controlled, and access limited).

2. Integrity of $CS_{acx}$ components, the security kernel and the objects connected thereto is controlled with the help of TSHM (interval $[o, t_0]$);

3. The security kernel is booted; integrity control, subject creation and access control procedures begin operating (at $t_0$ arrangement of IPE begins, in which time the only real subject – $s_{root}$ – exists in the CS [12]);

4. System components are further activated, system process subjects are created (interval $[t_0, t_3]$);

5. Stationary CS operation phase (before $login()$); user subjects are created (after $login()$; the interval is from $t_1$ and further).

Achievement of the absolute IPE in $CS_{acx}$ is ensured as follows:

– Launch (creation) of the SSM/OSM ($s_{acx}$) is guaranteed, and the objects connected thereto are controlled in terms of their integrity before activation with the help of hardware equipment (TSHM);

– $s_{acx}$ operates since early start-up of $CS_{acx}$ and implements the functions $Filter\_spawn_{acx}()$ and $Filter\_access_{acx}()$ immediately after initialization;

– After $s_{acx}$ is activated, there are only two subjects in $CS_{acx}$ – $s_{root}$ and $s_{acx}$, which are absolutely correct with respect to each other.

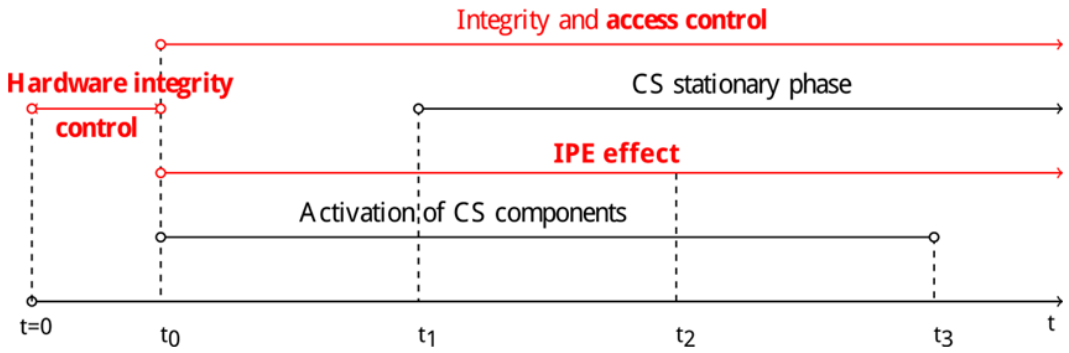


Figure 3: Schematic representation of $CS_{acx}$ operation stages with hardware integrity control, subject creation and access control by Accord-X.

Before the described start-up process, the set of allowed subjects in $CS_{acx}$ should be selected using the "soft operation mode" for the access control subsystem (the graph demonstrating creation of allowed subjects in one of GNU/Linux distributions is shown on Figure 4). This operation mode allows to "predetermine implementation of the initial CS operation phase" [4] in a way similar to that in the SO-model.

At least those objects listed in [13] should be used as controlled objects in TSHM. In case of controlling integrity of such objects and using the trusted start-up method for the bootloader and GNU/Linux, it becomes possible to ensure impossibility to start $CS_{acx}$ without switching on the corresponding security mechanisms of the access control subsystem.

It should be noted that the objects associated with the security kernel (especially information about the allowed subjects) play a key role in IPE designing, because if they are changed, the program environment can get "opened". Under such conditions, there arises a question about possible administration, i.e. possible emergence of the flow $Stream_{acx}(s, [s]_t) \to [s_{acx}]_t$ for some $s \in S_{acx}$.

This question in solved in $CS_{acx}$ by using the policy of absolute division of administrative and user authorities [4] described in the proof of Proposition 2.19. The security kernel of Accord-X operates at the level of the OS kernel (possesses maximum authorities together with the OS kernel); access to the objects associated with the security kernel is provided to the subject $s_{acx-admin}$, and access to the system-wide objects is provided to $s_{admin}$ (they possess a set of administrative authorities), with other subjects having no such authorities (they can only act to fulfill their functional responsibilities).

Most of the $CS_{acx}$ operation time, the administrative authorities are not used, with administration carried out only during special time periods (when there are no other subjects in the CS).

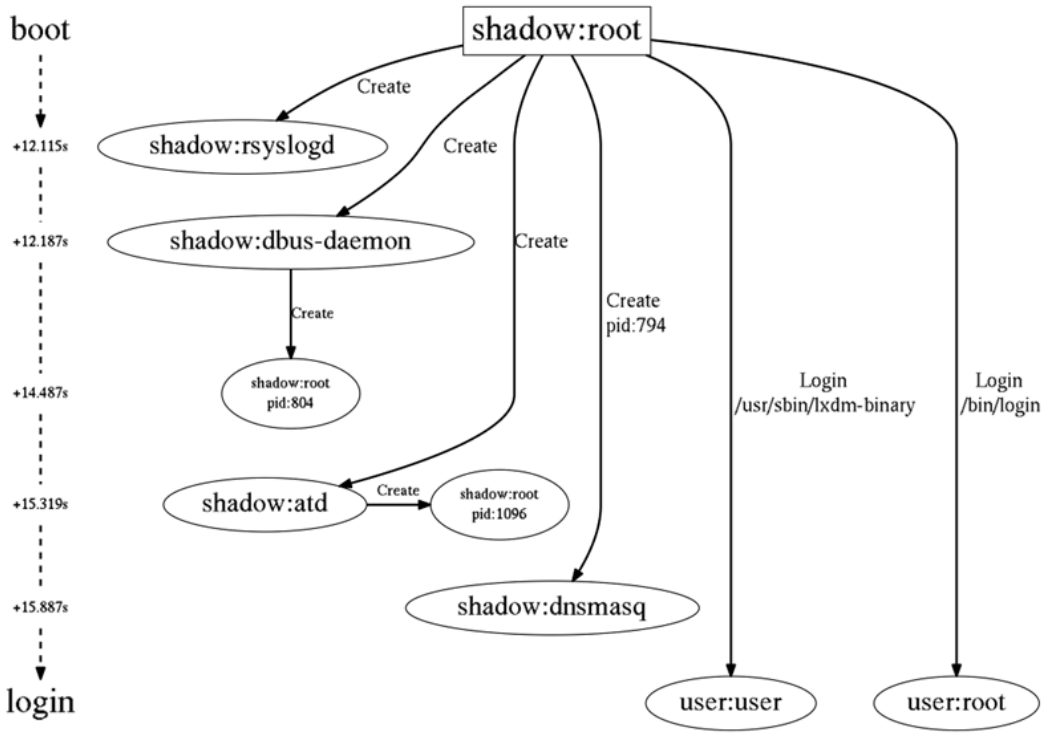


Figure 4: Graph demonstrating controlled creation of subjects in the process of Linux start-up.

### 3.2. Completeness of security mechanisms and correctness of several DST PUA interacting in GNU/Linux

Each particular GNU/Linux system can be unique: the system kernel can have unique configuration, various kernel versions can have different sets of system calls or their handlers, etc. On the one hand, standard DST cannot be applied to such unique distributions of GNU/Linux (or OS classes). On the other hand, all the distributions are more or less standardized in accordance with POSIX, SUS [10] and LSB [25], and transferability and completeness of security mechanisms can be ensured by providing for some variations.

The software and hardware complex "Accord-X" developed by the author is an embeddable or "complementary" DST PUA, which is universal and takes into account differences between Linux kernel versions 2.6 – 4.*, with necessary justification of completeness of the security mechanisms used therein.

Most of the security models fix subjects and objects. However, in modern CS, such a limitation seems artificial and difficult to implement – at least new objects emerge in the process of work (temporary files or objects in the memory having no file object). Subjects should be limited to ensure impossibility of unauthorized emergence of new subjects in the OS.

To limit the number of objects, the soft operation mode of Accord-X should be

used, which can help to set the rights in accordance with the least privilege principle [26]. To minimize the required adjustments of access rights (which are inevitable), the access control subsystem can be adapted, made more intellectual or even "self-learning" (adaptive), able to fix emergence of new objects in the OS (for example, those loaded through the network and saved in the file system). Moreover, object lists [24] and access rights can be replenished.

Completeness of security mechanisms in terms of control over all the available subject-to-object access paths in OS GNU/Linux is justified in Accord-X by embedding mechanisms [13, 17] and based on completeness of handler functions of Linux Security Modules (LSM) and system calls [17]. All possible subject-to-object access should be controlled, including *read*, *write*, *execute*, *create/mknode*, *delete/unlink*, *rename*, *link*, *mkdir*, *rmdir*, *cd*. However, in various Linux kernel versions the set of such handler functions, as well as the used parameters can vary, therefore, compatibility and certainty of all the implemented functions, as well as unification for any supported handlers' set, should be ensured in the access control subsystem. In this case, the number of access attributes, which can be set in the DST PUA, makes no difference (most of the operations can be described with the help of basic attributes *read*, *write*, *execute*).

Apart from confidentiality issues, the issues of information integrity are considered in Accord-X. Instead of combining confidentiality models with integrity models [19, 23], additional control over information integrity in case of any attempts to access the controlled objects is used. This means that objects' integrity is controlled, and making of unauthorized changes therein is fixed in the log of the access control subsystem.

Correctness of the access control subsystem in terms of combining several security policies is ensured by the peculiar way of embedding security mechanisms [13, 17]. Any default security mechanisms in GNU/Linux operate directly before or after all the policies in the access control subsystem. That is why, a CS with such a subsystem can be considered as a series connection of two finite automata [4] with one-sided interlocking in both cases [6]. Thus, all the default and embeddable access control mechanisms in GNU/Linux do not influence each other.

## 4. Conclusion

Thus, the article considers disadvantages of existing formal security models and selects a model, which is the most suitable for application in GNU/Linux and guarantees security of the information being stored and processed therein (SO-model of IPE).

It should be noted that the software and hardware complex "Accord-X" proposed by the author is homomorphous to the SO-model of IPE and meets the Basic Theorem of IPE, if the access and integrity control rights are correspondingly set. The security mechanisms in the access control subsystem of the considered DST PUA, together with TSHM, allow to achieve TCE ensuring and controlling boot of the CS security kernel. In addition to the SO-model, the author of the article considers the possibility to administer the DST PUA (with the help of absolute division of administrative and user authorities), the possibility to set access rights in accordance with the least privilege principle, the

possibility to account new objects in the CS, as well as the possibility to ensure information integrity. Thus, on the one hand, the access control subsystem being examined does not contradict the formal SO-model of IPE and does not conflict with the existing DST (it operates independently, supplements them), and on the other hand, it helps to solve all the questions listed in paragraph 2.1, which have remained open for formal models. At the same time, it ensures the required and sufficient security level both, for information (objects) and user environment, in which information is processed in GNU/Linux.

Further research on this topic can be connected with unification of the constructed mathematical model and creation of a meta model of a DST PUA, which would be applicable to various security tools and OS (Windows, GNU/Linux, etc.) in its initial form. If uniform rules of the access control subsystem are developed, it will be possible to create an access rights control subsystem, which will be common for various systems. On its basis, it will be possible to create various tools of remote centralized management over several types of access control subsystems at once, with their correctness and correspondence to formal security models in the CS guaranteed.

## References

[1] Bell, D. E., & LaPadula, L. J. (1976). *Secure Computer Systems: Unified Exposition and Multics Interpretation*. Bedford, Mass.: MITRE. MTR-2997.

[2] Castano, S., Fugini, M. G., Martella, G., & Samarati P. (1995). *Database security*. New York, N.Y.: ACM Press/Addison-Wesley.

[3] Denning, D. (1976). A Lattice Model of Secure Information Flow. *Communications of the ACM*. 19 (5), 236–243. DOI:10.1145/360051.360056.

[4] Devyanin, P. N. (2013). *Modeli bezopasnosti komputernykh sistem. Upravlenie dostupom i informatsionnymi potokami* [*Computer Systems Security Models: Access Control and Information Flows*]. Moscow: Goryachaya liniya – Telekom. (in Russian).

[5] Devyanin, P. N., Mikhalsky, O.O., Pravikov, D.I., & Scherbakov, A.Yu. (2000). *Teoreticheskie osnovy komputernoy bezopasnosti* [*Theoretical Fundamentals of Computer Security*]. Moscow: Radio i svyaz. (in Russian).

[6] Golovinskii, I. A. (2009). Blocking of Group Automata. I. One-sided Interlocking. *Journal of Computer and Systems Sciences International*, 48(1), 45–69. DOI:10.1134/S1064230709010055.

[7] Grusho, A.A., & Timonina, E.E. (1996). *Teoreticheskie osnovy zaschity informatsii* [*Theoretical Fundamentals of Information Security*]. Moscow: Yahtsmen. (in Russian).

[8] Harrison, M., Ruzzo, W., & Ullman, J. (1976). Protection in Operating Systems. *Communications of the ACM*, 19(8), 461–471. DOI:10.1145/360303.360333.

[9] Hoffman, L. (1977). *Modern Methods for Computer Security and Privacy*. Englewood Cliffs, N.J.: Prentice-Hall.

[10] *IEEE*. (2013). *IEEE Std 1003.1-2013. Standard for Information Technology: Portable Operating System Interface (POSIX): Base Specifications, Issue 7*. The Open Group, IEEE. Retrieved December 20, 2015, from http://pubs.opengroup.org/onlinepubs/9699919799/.

[11] *ISO/IEC*. (2009). *ISO International Standard ISO/IEC 15408-1:2009 – Information technology – Security techniques – Evaluation criteria for IT security – Part 1: Introduction and general model* (3rd edition). Geneva, Switzerland: International Organization for Standardization (ISO). Retrieved December 20, 2015, from www.iso.org/iso/home/store/catalogue_ics/catalogue_detail_ics.htm?csnumber=50341.

[12] Kanner, A.M. (2014). Linux: Process Life Cycle and Access Control. *Information Security Questions*, (4), 37–40. (in Russian).

[13] Kanner, A.M., & Ukhlinov, L.M. (2012). Access control in GNU/Linux. *Information Security Questions*, (3), 35–38. (in Russian).

[14] Konyavsky, V.A. (1999). *Upravlenie zaschitoy informatsii na baze SZI NSD* "*Accord*" [*Information Security Control based on DST PUA* "*Accord*"]. Moscow: Radio i svyaz. (in Russian).

[15] Konyavsky, V.A. (2005). *Metody i apparatnye sredstva zaschity informatsionnykh tekhnology elektronnogo dokumentooborota* [*Methods and Hardware Tools for Protecting Information Technologies of Electronic Document Management*] (Doctoral dissertation). National Research Nuclear University MEPhI, Moscow, Russian Federation. (in Russian).

[16] Malcev, A. I. (1970). *Algorithms and Recursive Functions*, Groningen: Wolters-Noordhoff Pub.

[17] Matveychikov, I.V. (2014). Overview of Dynamic Operating System's Kernel Hooking Methods (Study Case of Linux Kernel). *Bezopasnost Informatsionnykh Tekhnology*, (4), 75–82. (in Russian).

[18] McLean, J. (1990). Security Models and Information Flow. *Proceedings. 1990 IEEE Computer Society Symposium on Research in Security and Privacy* (pp. 180–187). Oakland, CA, USA: IEEE. DOI:10.1109/RISP.1990.63849.

[19] Rakitskiy, Y.S., & Belim S.V. (2011). Model of Union Two Mandatory Security Policies. *Bezopasnost Informatsionnykh Tekhnology*, (1), 125–126. (in Russian).

[20] Sandhu, R., Ferraiolo, D.F., & Kuhn, R. (2000). The NIST Model for Role-based Access Control. *Proceedings of the fifth ACM Workshop on Role-based Access Control* (pp. 47–63). New York, USA: ACM. DOI:10.1145/344287.344301.

[21] Scherbakov, A.Yu. (2001). *Vvedenie v teoriyu i praktiku komputernoy bezopasnosti* [*Introduction to Theory and Practice of Computer Security*]. Moscow: S.V. Molgacheva. (in Russian).

[22] Scherbakov, A.Yu. (2009). *Sovremennaya komputernaya bezopasnost* [*Modern Computer Security*]. Moscow: Knizhny mir. (in Russian).

[23] Shcheglov, K.A., & Shcheglov, A.Yu. (2014). A Consistent Model of Mandatory Access Control. *Journal of Instrument Engineering*, 57(4), 12–15. (in Russian).

[24] Shcheglov, K.A., & Shcheglov, A.Yu. (2015). New Approach to Data Securing in Information System. *Journal of Instrument Engineering*, 58(3), 157–166. DOI:10.17586/0021-3454-2015-58-3-157-166. (in Russian).

[25] *The Free Standards Group*. (2015) *Linux Standard Base (LSB) 5.0*. The Linux Foundation. Retrieved December 20, 2015, from www.linuxfoundation.org/collaborate/workgroups/lsb

[26] *[TCSEC] Trusted Computer System Evaluation Criteria.* (1985). Ft. Meade, Md.: Dept. of Defense, Computer Security Center. DOD 5200.28-STD (supersedes CSC-STD-001-83).

[27] Ukhlinov, L.M. (1996). *Upravlenie bezopasnostyu informatsii v avtomatizirovannykh sistemakh* [*Information Security Control in Automated Systems*]. Moscow: MEPhI. (in Russian).

[28] Zegzhda, D.P. (2002). *Printsipy i metody sozdaniya zaschischennykh sistem obrabotki informatsii* [*Principles and Methods for Establishing Secure Data Processing Systems*] (Doctoral dissertation). St. Petersburg Polytechnic University, Saint-Petersburg, Russian Federation. (in Russian).

**Note**

**1.** The "zero" element is further used to formally describe the prohibition of emergence of an information flow to an object and the prohibition of creating a subject in the system.