Information Security Management in Heterogeneous Systems - Identifying the challenges

Paul Lackner
paul@lithilion.at

ABSTRACT

This paper discusses security from the perspective of heterogeneity, which so far has not been in the focus of security research. After carrying out the state of the art analysis conforming this position, this phenomenon is further discussed. However, one of the major conclusions derived by the authors is that a secure heterogeneous environment is an achievable goal. Existing services suffer from the problem of not having a common definition on how to communicate with and trust each other, as well as how to choose the right service for a specific requirement or action. To address this gap, the paper will focus on specifying the requirements for common, standardised interfaces for specific layers. These requirements are intended to be used for supporting the development of a more secure, diverse system landscape. The paper then continues with discussing potential use cases for the requirements and then presents still open challenges that need a scientific solution. The major conclusions drawn from the state of the art analysis are again summarized in the final chapter of this paper.

CCS CONCEPTS

• **Security and privacy → Systems security**; • **Computer systems organization → Architectures**; *Distributed architectures*; • **Information systems**;

KEYWORDS

heterogeneity, interfaces, data exchange, federation, standardisation

1 INTRODUCTION

As shown in the literature [18][4] and demonstrated by very worrying practical examples [16], the growing heterogeneity of system landscapes has resulted in whole industry sectors becoming vulnerable to cyber attacks[1]. This vulnerability is caused primarily by architecture considerations not being paid enough attention to. The problem relates back to the so called "organic growth" of infrastructure and applications, which system developers traditionally try to overcome with connecting new systems by so called pipes. The ensuing lack of an architectural model is reflected in today's information security concepts. Trying to create a homogeneous system landscape, to mitigate classical pitfalls created by heterogeneous systems, is also a non achievable goal, as presented in the paper. However, had the problem be looked at from an interface perspective in the first place, some of the worst security holes caused by these "pipes" might have been avoided. That is why, drawing on the lessons learned from leading reports [16], this paper focuses on the need for an architectural model that is building on an interface oriented perspective. Starting with an overview of the current situation and existing approaches, this paper first describes the challenges and gaps which need to be addressed. Following the problem description, a further analysis of vulnerabilities is carried out. From the results of this analysis the paper derives a set of requirements that are necessary for guiding the construction of an architectural model capable of growing with a companies Information Technology (IT) infrastructure. The final conclusions summarised in the last chapter of the paper will serve as orientation for setting the aims the planned architecture model one has to achieve. The practical relevance of providing such a scalable and adaptable architectural model is expected to immediately show through a significant reduction of attack surfaces. With this goal being achieved, it should be possible to focus the defensive measures on a much smaller number of issues. Besides the organisational and technical benefits, less effort needed should also help to reduce the cost.

Apart from these practical improvements, the scientific contribution of the approach will also have significant impact, because once the lack of architectural framework and model are overcome, new ways of incorporating cyber security aspects in system designing and development will be possible.

2 SECURITY, HETEROGENEOUS AND HOMOGENEOUS SYSTEMS

To fully understand the correlation between information security and various system architectures it is necessary to understand the correlation of *Security* and *Heterogeneous Systems* which will get explained in the following sections.

2.1 Core Characteristics of Heterogeneous Systems

Finding definitions of homogeneous or heterogeneous systems in terms of IT is surprisingly difficult. One definition of a heterogeneous system was found in the context of software development: "Allow distinct computational models to coexist seamlessly [...]" [2]. Therefore homogeneous systems tend to be kindred computational models. Heterogeneous systems are also described as a mixture of various Operation System (OS), software stacks and libraries (microservice context) [14], which can also be extended to various hardware components, which gained attention after the disclosure of the first hardware based major-bugs Spectre [12] and Meltdown [15].

A more complex definition is found in the context of IT service value networks: "Heterogeneity [...] can be defined as the diversity and alterity of the attributes of the summed applications, platforms, infrastructures, actors, technologies, interfaces, and tools [...]" [8]. These are the elements where heterogeneity may occur through various attributes. Table 1 shows an example matrix of the collaboration of the various elements and attributes to create heterogeneity.

[1] https://dirtypipe.cm4all.com/

		Elements						
Attributes	Triad characteristic	Applications	Platforms	Infrastructures	Actors	Technologies	Interfaces	Tools
Pricing policy	-				x			
Service Level	A	x	x	x				
Standardisation	CIA	x	x	x	x	x	x	x
Interoperability	CIA	x	x	x			x	
Performance	A			x				
Communication	CI					x	x	
Version	CIA	x	x	x			x	x
Technology level	CIA		x	x		x		

Table 1: An extract of the concept matrix to show the interactions of practically highly relevant attributes with different elements to toggle heterogeneity [8]. The columns represent the elements where heterogeneity may occur through various attributes (represented by the rows). Attributes present in an element are marked with an "x". This extract focuses on the element interface and shows the impact of the attributes standardisation and its relative interoperability. It is enhanced by the CIA-triad characteristic and shows the contribution of each attributes to the triad. Therefore the triad characteristic for an attribute applies to all intersections of that attribute with an element.

Diversity is described as the number of different elements, while alterity describes the differences between two elements. The alterity between a Windows and Ubuntu system is much higher, than between a Debian and Ubuntu system. They also identified various attributes which are differently matched to the respective enumerated elements. The elements *application/software*, *platform*, and *infrastructure* can be seen equal to their "as a service" cloud pendant. *Actors* are all participating members in a service network. *Technologies* are meant as the communication methods, *interfaces* as the offered possibilities of a service (Application Programming Interface (API), protocols), and *tools* as the tools to use and manage the service.

Another paper generically describes heterogeneity as a statistical property for the measurement of the diversity of given characteristics in IT systems [28]. It is proposed that heterogeneity may be calculated through the Herfindahl-Hirschman-Index (HHI) [9] or Entropy Measure [10]. Both indexes are used in economics as well as in biology to calculate and measure diversity of firms and the bio ecosystem. The HHI indicates a diverse ecosystem by resulting in a low index, while the Entropy Measure results in a high index. Furthermore they define, that the more different values for a distinctive attribute are, as well as the higher the disparity of these values are, the higher is heterogeneity.

Good heterogeneity is a wanted, controlled, and understood mixture of various hardware and software stacks, while bad heterogeneity is the mixture of various hardware or software versions due to the lack of a working substitution and update process.

2.2 Security in Heterogeneous Systems

To apply security to a system, all elements of that system need to be in focus. Table 1 shows an extract of attributes and elements where heterogeneity may occur in a system. This means that these attributes and elements are present in systems, therefore they need to get addressed in security considerations. As this paper already showed, heterogeneity is measured as a statistical value with a lot of elements and attributes to consider, which means the chances of finding a real homogeneous system is very low. Also, if comparing homogeneous systems and heterogeneous systems, the only difference is that the elements and attributes have the exact same value at homogeneous systems, and differ in heterogeneous systems. Therefore, to simplify security considerations, it is proposed to always consider all elements and attributes as present and given in a system. Standardisation of shared infrastructure always helps in developing heterogeneous systems [18]. Table 1 also shows the contribution of each attribute to the CIA-triad. This paper further aims to simplify security definitions as well as any other interoperation definition by proposing the definition of common interfaces, the most crucial part of heterogeneous systems.

2.3 Understanding Interface Structure

An API is required for communication between different applications and application parts. It is used to exchange data and instructions between different computational parts. An API defines the name of the interface as well as the required parameters and its output. Some programming concepts refer to interfaces for simple functions (i.e.: HTTP APIs), whereas other concepts (Object-Oriented Programming (OOP)) refer to interfaces as simple class definitions, which defines class input parameters and functions of a class. An interface is mostly described as a formal definition of a desired action which still needs to be implemented. The JAVA documentation refers to interfaces as a contract between the implementing code and the outside world[2]. To communicate with another part of a software, the interface needs to get published, while the implementing code is not.

IBM lists the following main reasons for using APIs[3]:

- *Collaboration & Innovation*: APIs, especially open APIs, trigger a vast collaboration process, which enables anyone to build applications around your application and extend your ecosystem or integrate your application into their ecosystem. Either way your application will experience a higher usage and acceptance.
- *Security*: Providing access to computing resources and data over an API means to establish access control to these resources. Authentication, Authorisation, Encryption and other forms of security measures can be applied to secure communication between services.

[2]https://docs.oracle.com/javase/tutorial/java/concepts/interface.html
[3]https://www.ibm.com/cloud/learn/api

- *Monetizing*: The forms of access controls described in the security bullet point also enables to limit access to specific paying parties. Therefore, providing services and data can generate an income for offering companies.

Another source lists advantages and disadvantages of interfaces and their corresponding APIs [23]:

- *Stability*: Interfaces provide stability to dependent application through an abstract layer above the corresponding application.
- *Modularisation* An interface does not reflect the complexity of its corresponding implementation but is only an abstract description of a given method or function. A developer of an application using such interface therefore has no need of understanding the implementation.
- *Reusage*: An API enables, similar to a programming library or function, its simple reusage.
- *Interoperability*: Application interfaces lack the possibility of language interoperability. This is problem is addressed at API level with wrapper protocols (e.g.: Hypertext Transport Protocol (HTTP)).
- *Changeability*: Interfaces are very hard to change at the moment they are used. Therefore interfaces are versioned. Obsolete interfaces are marked as deprecated and to be removed in an upcoming version, but changing an interface is nearly impossible.

This paper will introduce a new approach that utilises the advantages of already established interfaces as described above and tries to remove the need of a wrapper and implement interoperability into the given interface.

A graphical interface must be designed very clearly. If a user does not know, what to do next, there is a problem in the interface design [25]. This can be used in a developer/programmatic way as well. If a programmatic interface is not well designed, a developer may misuse the interface or implement it wrongly and therefore gather wrong outcomes.

Interfaces and APIs are similar, but different concepts. An interface only describes a wanted command, an API however consists of an interface and its implementation. As this paper solely concentrates on interfaces, it sometimes references on API rules and best practices, as long as these practices relate on the interface part of the API.

2.4 Inter-Process Communication

Communication between processes and application is an essential asset in IT. There are some major communication methods defined[4][5]:

- (Berkeley) Sockets: Berkeley sockets are a de-facto standard to establish network communication. Sockets work with any underlying major network protocol.
- Pipe: A pipe is a unidirectional read and write channel. The read and the write head may be on different positions. To establish a bidirectional process communication at least two pipes are required.
- Message Queue: A message queue is like a software implementation of a BUS system. Messages are committed to it, and chosen recipients are enabled to subscribe to these messages and answer to them.
- File: A file is being exchanged. This might happen via disk access on the same machine or via a network and therefore probably a socket.
- Memory Sharing: Multiple programs on the same machine have access to a shared memory space (RAM), where they can read and write data and commands for each other.
- DMA-Infiniband: Infiniband is a hard- and software network connection family, which is known for high-troughput and low latency and a de-facto standard for communication in super computers. It is an alternative protocol to ethernet, but supports Ethernet-over-Infiniband for compatibility reasons. Infiniband connections are similar to sockets.

Sockets are a major inter-process communication (IPC) component, working over a network (and via *localhost* also on a local machine). With the arise of microservice development and intensive cloud usage, the usage of sockets for IPC also rises [14] compared to the other methods. Pipes are only used locally on OS level but need a socket to communicate over a network. As described earlier, pipes are error prone, due to their non standardisation and are a non desired future solution. Message queues are implemented via an application on OS level which also need a socket for network communication. File sharing is completely dependent on OS level file behaviour and sockets for network communication. File sharing is considered non-performing as multiple components are involved. Memory Sharing is a very performing way of IPC, but also considered insecure, because various processes are granted access to the same memory space, which may enable memory corruption, data exfiltration, etc.

Sockets are the only IPC method that work independently on a local machine and over the network. They also have an already standardised interface to communicate with, are build for modular usage, therefore provide a level of abstraction, which makes them compatible with most applications, and are already used widely. Berkeley sockets are the optimal component to enhance our desired API standardisation approach as described in section 4. Infiniband connections may replace sockets in a future version, but lack the level of real world distribution compared to sockets.

3 CHALLENGES ASSOCIATED WITH INFORMATION SECURITY MANAGEMENT IN HETEROGENEOUS SYSTEMS

IT services are defined as intangible, perishable, and heterogeneous [11]. The heterogeneity rises when more complexity in the IT service is required [28]. A challenge to establish security in heterogeneous systems is the vast amount of combinations of elements and attributes to consider and their endless variations, which was also stated by the USENIX conference: "Nevertheless, the cloud and serverless platforms are now facing a problem that operating systems had before POSIX: their APIs are fragmented and platform-specific, making it hard to write portable applications"[6]. A reduction

[4]https://docs.microsoft.com/en-us/windows/win32/ipc/interprocess-communications

[5]https://en.wikipedia.org/wiki/Inter-process_communication

[6]https://www.usenix.org/publications/loginonline/transcending-posix-end-era

of complexity by reducing the variations of the right elements will help. Therefore, this paper proposes to standardise the element *interface* to reduce its variations and reduce complexity in this layer which will be introduced in the next section. When standardising the element interface to a common specification, all other elements could easily grow in heterogeneity. When having a low heterogeneity in the element interface, there can be a high heterogeneity in all other elements, while when there is a high heterogeneity in the element interface, achieving a high functioning and secure heterogeneity in all other elements seems unrealistic and highly complex.

3.1 Open Challenges that Demand Scientific Solutions

Heterogeneous, distributed systems rely on trust and reputation of the respective services. Services just rely on other services that they act security and privacy preserving as described above by simply trusting them [26]. This is a huge issue as various services are operated by different operators, vendors, etc. where service providers often do not know with whom data is shared and what is done with if (e.g.: distributed logins via SAML and OIDC). At the moment, if data is transferred to another service or user, one renounces the sovereignty of the data to this particular service or user. This party is technically able to do whatever it wants to with the data. Trust decisions and delivery of reputation to distinct services are a major unsolved challenge [18]. Relying on the existence of service level agreements or a centralised trust entity such as a PKI is not realistic or does not represent the purpose of heterogeneous, distributed systems. Requesting services or systems have to decide on their own if another service is reliable and trustworthy. In order to do this, some orientations exists, but no common solution is standardised. This starts at the lack of a common definition of trust and reputation of services and continues to the different approaches to achieve the various goals that are not compatible with each other. There is no common understanding of the initial problem nor the goal that needs to be achieved; e.g.: There are different algorithms to establish trust and reputation for P2P file shares. They use PATROL-F [24] and ad-hoc networks use PTM [1]. Both try to reach the same goal - provide security to services via establishing trust and provide a reputation score to services, but have a different understanding of trust and reputation and are not compatible with each other.

When data is transferred or shared with another service this service needs to apply the same security and privacy policies to this data as the original service. A scientific solution to ensure a policy enforcement of a local system to a remote (third party) system and possible further remote systems is needed to increase the trust of users into the software products. Some concepts, that may help in finding a solution will be presented in the following paragraphs.

To formulate a problem description:

- How to guarantee seamless service interaction of various, different services.
- How to guarantee policy enforcement and rightful data usage on not-owned systems.

To solve these problems, two layers may be addressed: the interface and the data itself. Is it possible to solve the problem of policy enforcement and data usage on remote systems solely via interface descriptions?

A further problem goes with the expected federation of services. If services can seamlessly communicate with each other, one can assume that federation will take place. This also means that multi-hop federation can and will happen, therefore service A federates to service B which federates to service C. Through the seamless communication service A could communicate with service C through service B, without being directly connected, but if service B fails, there still is a need for service A to communicate with service C. An outage of service B must not cut of all services behind it [21]. Already existing, established solutions (Domain Name System (DNS), OSPF, BGP, Network Time Protocol (NTP), etc.) to provide data to federated systems are working great in their "security-version" to provide integrity but do not consider confidentiality.
Furthermore all federation problems will align to these services [3].

3.2 Definition of Common Interfaces

This paper introduces a way to define heterogeneity based on a layer system similar to the Open Systems Interconnection (OSI) layer or the TCP/IP layer, which is displayed in Figure 1. The first layer will consist of the characteristic hardware, while the third layer is the characteristic kernel. The fifth layer consists of the characteristic application. If the interfaces between these layers are standardised, heterogeneity in between the layers is achieved without a high complexity. A working example of this, would be the network layer of the OSI layer and its standardised protocols (Transport Control Protocol (TCP), User Datagram Protocol (UDP), HTTP, DNS, NTP, etc.). The protocols are standardised by an international organisation (e.g.: Internet Engineering Task Force (IETF)) and published. An application communicating with the network (e.g.: web server) implements the HTTP protocol to do this. There is no need to know what application/implementation (httpd, nginx, IIS, Firefox, Chromium, curl, etc.) this server is communicating with, as long as both of them agree to communicate over the same, standardised protocol.
The following standardised interfaces would be necessary:

- Hardware - Kernel
- Kernel - Application
- Application - Application
- Network

The layer "Hardware - Kernel" introduces an interface between the hardware and drivers as well as the system kernel. This interface will be the least complex interface of all four introduced ones, as the possible commands of a system towards the hardware are limited. The next interface "Kernel - Application" roughly introduces file and user management. System call interfaces will be defined here; therefore this interface is more complex. The most complex interface will be the "Application - Application" interface, as it introduces an interface for applications to communicate with other applications. The possibilities are endless, therefore the complexity is the highest.
The interface "Network" already exists in a loose way and is one of the reasons the internet as we know it today works with many different systems involved. It is seen here solely as a transport layer, where application interface calls are transported to another system.

The network interface only serves as a sub-interface alongside the other three interfaces.

This paper proposes to establish a common definition of interface regarding their naming, output values (especially status codes) and input parameters. The implementation of the interfaces in the various application is out of scope and will not be regarded as it remains an obligation to the developer of the application. The naming of these interfaces must align to a specific convention which allows to determine the purpose of the interface by simply knowing the name on the one side and on the other side allows the interface to be found when looking for it. The output values as well as the input parameters must meet common criteria. The requirements onto parameters of various interfaces share a lot of identical aspects and therefore parameters may be consolidated alongside various interfaces. Therefore, the author proposes to establish a tree-based interface architecture separated by the various introduced layers. Therefore, interfaces from the layer "Application - Application" are related to each other inside the layer, as well as interfaces from the layer "Hardware - Kernel" and "Kernel - Application". To create appropriate interfaces, one must understand their purpose and its requirements. In order to do this, this paper tries to categorise and group applications. To give an example: The application *MySQL* may get categorised as "database - relational"; the application *PostgreSQL* will get assigned to the same category. The application *MongoDB* will get assigned "database - non-relational - object store", while *Neo4j* will get "database - non-relational - graph". *MySQL* and *PostgreSQL* will use the exact same interfaces, as they are the same type of application and will share some interfaces with *MongoDB* and *Neo4j*. *MongoDB* and *Neo4j* will share more interfaces with each other as with the relational databases, but not all interfaces as they are more related to each other, but not of the same kind. All of these applications will also relate to a complete different application, like *postfix* ("Groupware - Messaging - Store and forward"), and therefore share some interfaces (e.g.: login interface) and/or parameters but not many.

Two frameworks exist to categorise applications and protocols: OSI and TCP/IP. They declare several layers which are needed in inter-application network communication. The OSI framework distinguishes between seven layers, while the TCP/IP framework only knows four. This is visualised in Figure 2. When comparing our defined layers of the heterogeneity framework with the OSI and TCP/IP framework, it gets clear that the usermode layer of heterogeneity matches with application layer of TCP/IP, the drivers & hardware layer of heterogeneity matches with the link layer of TCP/IP while the operation system layer of heterogeneity matches with the transport and internet layers of TCP/IP. In this case, the OSI layer provides unnecessary complexity, which is why the author will further work with the TCP/IP framework. Also, the border between usermode and operating system layer matches the border of the application and transport layer which is exactly where socket are placed. Therefore, sockets are a perfect match as a foundation for our modular interface concept for usermode applications.

3.3 Interoperability - Standardisation of Interfaces

A major challenge in providing heterogeneous systems is the establishment of full interoperability between systems. This challenge may be the most important challenge. This paper proposes to establish a deep understanding of various systems, and that there is practically no such thing as a homogeneous system. Therefore system engineering and security thoughts must focus towards heterogeneous systems and their interoperability. Interoperability strongly depends on standardising the element interface. Establishing interoperability means the integration of independent data sources and the processing and accessing of the data and results as if the distinctive systems were only one [4]. Heterogeneous systems have a need of interoperability which opens the local system to new vulnerabilities. Therefore a need of security in interoperability is identified. The potential loss of control and the fear of compromisation affects the implementation of shared infrastructure, while the loss of vendor lock-ins affects the development of interoperable systems [20]. A new European regulation[7] may increase the future development of shared, interoperable systems and therefore increase the need of security in heterogeneous systems. As already shown in the history of gaming, the definition of common, standardised interfaces increased the usage of said systems and increased the ecosystem around these applications. This can be seen at the popularity of DirectX at game development[8] as well as the increased usage of Linux systems in gaming after the creation of a compatibility layer[9]. The requirements for interoperability are transparency, autonomy, and security [4]. Users and programs should access resources in a uniform way. They must be permitted to access resources they have access to on the local system and they must be denied access to resources they cannot access on local systems. Therefore local Access Control List (ACL) must also be enforced on shared systems. The systems must be security preserving. A description of privacy preserving microservices has been made [27] which can also be applied to security. They define microservices privacy preserving when it only process data they are entitled to and only share data with services that are entitled to process this data. This definition can be adapted to security: A security preserving service only allows local and remote users and programs to access data they are entitled to on local systems and only processes remote access requests from local and remote users to remote systems to data they are entitled to. Of course the privacy preserving concept also applies to security, therefore a security preserving service also implements the privacy preserving principles. The main weakness of this definition is the shift of responsibility to a receiving service and therefore it relies on the trust of the other service instead of an enforcement of a policy. Especially in a microservice environment with a zero-trust security concept this contains a certain irony. The challenges identified in this section of the paper need to be properly addressed at the interface level to make full use of approaches developed in leading research projects in the fields of database systems [19][22][6] and networks. Against

[7] https://ec.europa.eu/info/strategy/priorities-2019-2024/europe-fit-digital-age/digital-markets-act-ensuring-fair-and-open-digital-markets_en
[8] https://www.pcgamer.com/history-direct-x-windows-microsoft/
[9] https://www.slashgear.com/steam-proton-has-opened-the-gaming-floodgates-for-linux-users-22617714/

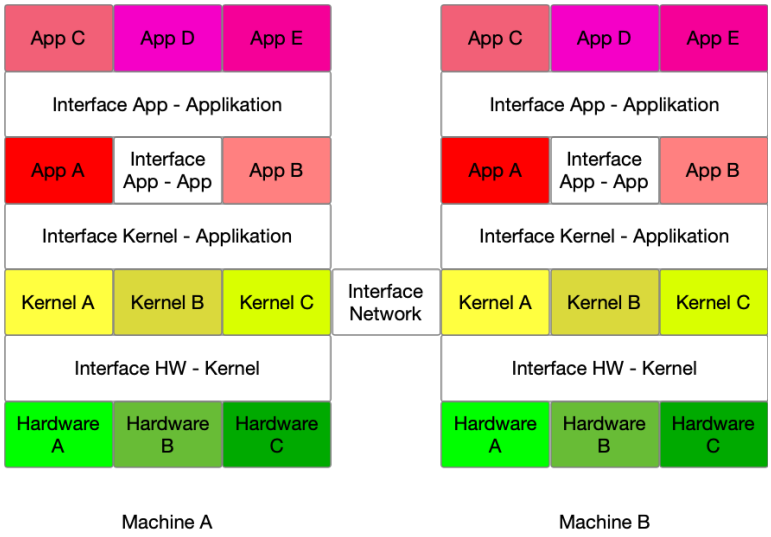

Figure 1: The introduced layers of heterogeneity. The first layer represents the hardware layer, the third layer represents the system layer, and the fifth layer represents the application layer. Between these layers is the element interface. If a standardised interface between the various layers exists, heterogeneity in a specific layer can be achieved without a high complexity. Multiple kernel in the same OS may seem possible.

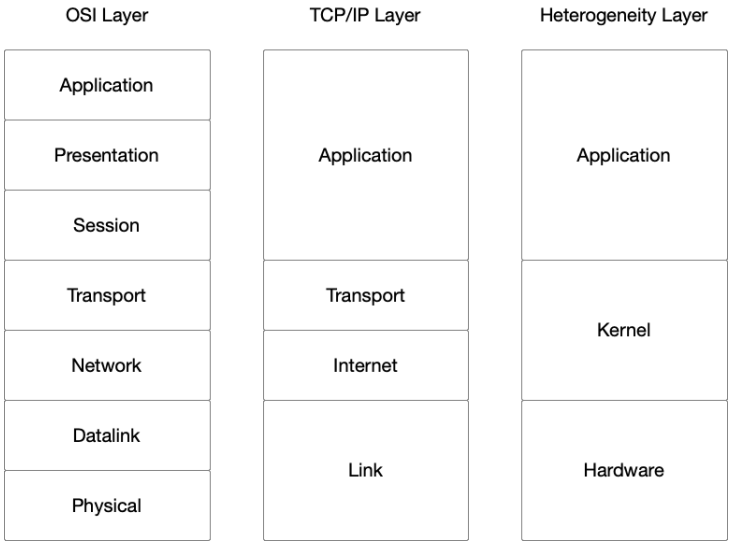

Figure 2: A comparison of the OSI and TCP/IP concept. The TCP/IP concept unifies in the application layer the application, presentation, and session layer of the OSI framework, as well as the datalink and physical layer into the link layer. Moreover these layers can also be matched with the heterogeneity framework concept. The TCP/IP application layer represents any usermode application, while the transport and internet layer is represented in the kernel layer. The link layer includes driver software and the corresponding hardware.

this background the author will focus on developing a solution for providing an interface driven perspective on information security management in a heterogeneous system landscape. This paper will focus on the application layer. Also, it will focus on general applications and exclude any high-performance applications, which have special requirements.

4 INTERFACE STANDARDISATION

This paper will introduce our interface standardisation approach via presenting it on the application layer. It will focus on general applications and exclude any high-performance applications, which have special requirements. Each standardised interface has a standardised defined structure concerning input, output and naming.

To establish a secure standard, it is highly recommended to create easy-to-use interfaces with a low complexity and no specifications that are double defined (cf. heartbleed vulnerability [10]).

4.1 Application Structure

This paper introduces a comprehensive approach to standardise interfaces. It will tackle an interface naming convention as well as the basic structures of interfaces, therefore the author standardises the required and optional input parameters and result of interfaces. Our approach is based on the structuring of interfaces based on the heterogeneity framework structure as well as the nature of every interface. The presented structure is based on an (inverted) tree structure, where lower levels inherit every property from above levels. It starts at a root level where any properties are defined that apply to all application layered interfaces. This paper starts defining at the application level. The root level is out of scope and may get filled with berkeley sockets. Every aspect above the root level will be defined by the sockets, so there is no need to define them. Below this level the levels are simply enumerated. Level 1 distinguishes between any raw kind of software type (i.e.: Database, Groupware, ERP system, etc.), whereas level 2 distinguishes these types into more detailed ones (e.g.: relational and non-relational database), and so forth. Any application will be defined to be able to get categorised through this approach to at least level 1 but may be categorised to any enumerated level below. This structure is presented at Figure 3.

To illustrate the concept this paper describes an example with instant messengers. Every instant messenger mainly consist of the same main components; texting, voice and video calls, and file sharing, in 1-1 and 1-n chats. Logging into a messenger will probably work via an API on the application level (every login to a system rather has the same characteristics), while the other API calls will probably be called on level 3 in the category instant messaging (see Figure 3) or lower. Therefore it will be completely irrelevant which instant messaging implementation is used, as they all are implementing the same standardised interfaces.

A similar structure may be possible for the hardware (Figure 4) and kernel (Figure 5) layer. The author did not sketch a network layer, as he believes the presented structure for the application layer includes every networking aspect and therefore a network layer will be expendable.

4.2 Interface Architecture

Sockets have been identified to provide a solid base for the proposed interface architecure for inter-process communication. Sockets enable to act platform agnostic and don't require the caller to create different looking requests for local and remote systems. In combination with a standardised API (standardised naming, input and output), no Object Request Broker (ORB) or similar is required. Therefore no central component is required in distributed systems is needed [5]. This enables multiple opportunities for further designing the architecture:

- Building on top of raw sockets - Building a new protocol on top of raw sockets would have the big advantage of having no dependencies except the socket itself. It would enable the usage of every transport protocol possible. Building on top of raw sockets also has the big disadvantage of creating a new protocol[11], therefore every aspect of every established protocol needs to be reinvented and re-implemented.
- Building on top of HTTP - Building the new standard on top of HTTP will lead to a new perspective. The API would be able to use URLs and HTTP Methods to address actions very specific. Also, such an approach would meet the needs of the modern "microservice style" of programming.
- Building on top of a Message Queue
- Building on top of an Enterprise Service Bus

4.3 Interface Naming Convention

Standardised interfaces need to be used to be effective. Therefore, they need to be found. Each person using interfaces must be able to find its respective interface, thus standardised interfaces need to be named according to a naming convention. Also, an interface must be easy understandable for any developer to easy find and implement [23]. An effective measurement is to name interfaces in an easy understandable, expressive way (e.g.: "drop", "delete", "remove" for a delete operation) and to spell them correctly. A name for an interface needs to express exactly the interfaces purpose but should be as short as possible. The same naming best practices apply to parameters.

The desired naming convention will strongly address the described tree structure (displayed in Figure 3). Each level will be presented in the naming bottom to top with the desired action. A login action interface to a graph database will be called *login* possibly, where as a login action interface to a relational database may also be called *login*, as this is a universal interface. An "addEvent" action to a calendar application may look like *addEvent_calendar_groupware*, where as a "send mail" action may look like *sendMail_storeAndForward_messaging_groupware*. The first described interface starts at level 2 while the second described interface starts at level 3.

4.4 Security and Privacy of Interfaces and Data

Standardised interfaces grant the possibility to establish a standardised set of security measurements. This may include mandatory data encryption through only accepting encrypted data, encryption algorithm enforcement (compare TLS (TLS) handshake), and mandatory key and permission management. These measurements may be a way to ensure policy enforcement of data on remote, non-owned processes but are still in the very beginning of a draft.

4.5 Adoption and Applications

Establishing a standardised set of APIs creates a lot of possibilities to ease the process of creating and testing applications, as well as making the interoperable. Establishing a standard also enables to establish standardised security. Some opinions state, that standardisation slows innovation, but the author states that this depends on the implementation and scope of the standard. Therefore, the paper will propose possible future applications which get possible with the standard and a possible implementation and adoption process for the market.

[10] https://www.cisa.gov/uscert/ncas/alerts/TA14-098A

[11] https://xkcd.com/927/

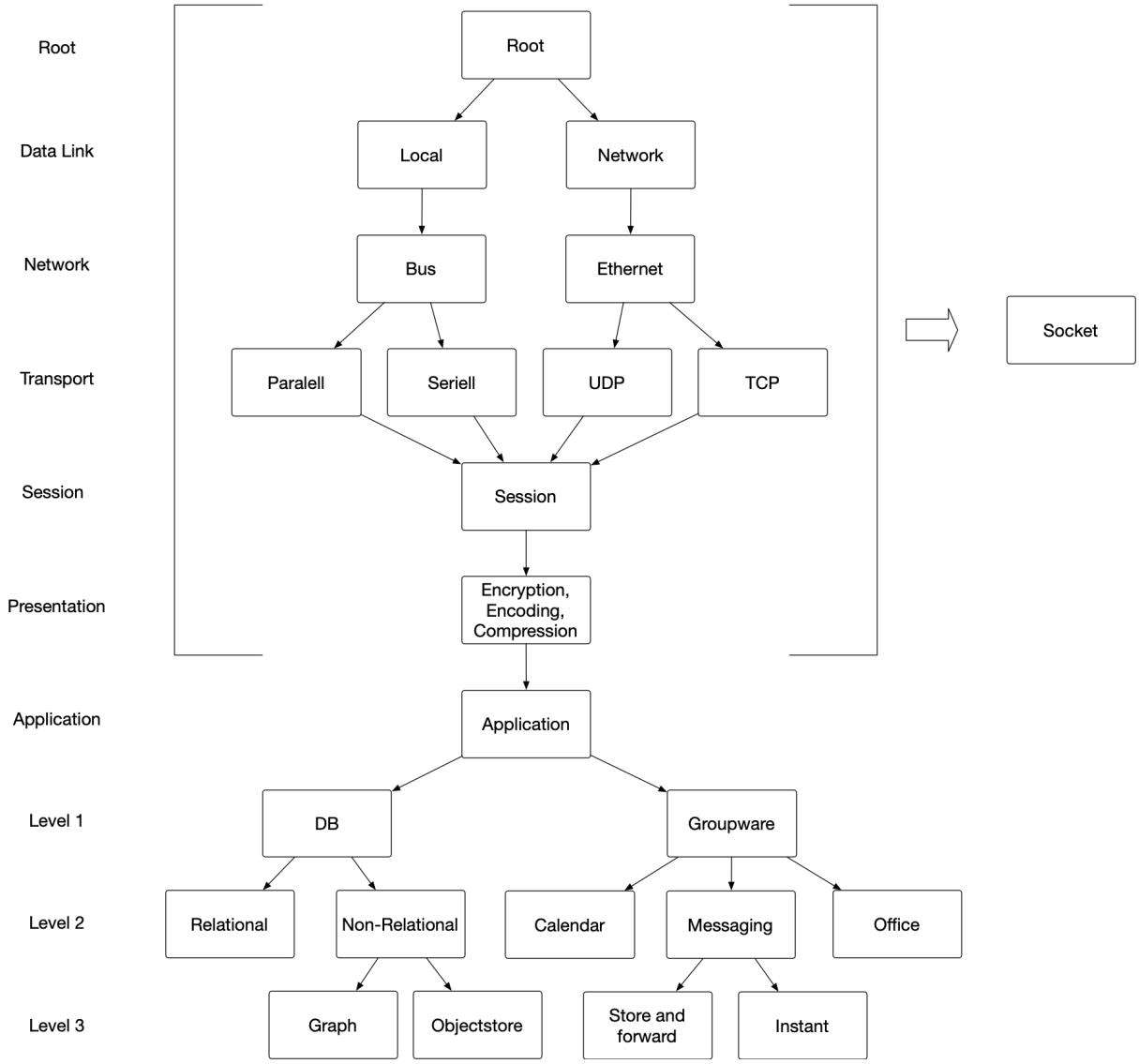

Figure 3: Interface structure for application based interfaces. Every interface is structured based on the heterogeneity framework and TCP/IP framework and further defined through their distinctive property. The tree structure requires every property defined in an upper level to be reflected to all lower levels. This image only shows a small extract of a possible interface structure and may be expanded on any level except the root level. The root level may be equal to berkley sockets.

4.5.1 Standardised test-driven development. A standardised set of APIs enables a complete new way and boost of Test-Driven-Development (TDD). When designing an application, a set of required (standardised) APIs will be identified. As the naming and in- and output of the API is standardised, test could also be standardised, which means tests could be created by someone else and sold to the market. If specialised companies create tests and sell them, it will increase quality of the tests and reduce the resource usage for test creation to a monetary problem for software manufacturers.

4.5.2 Standardised Security Pentesting. A standardised set of APIs also enables a standardised security penetration testing of applications. The same principle as with the standardised tests applies to the penetration test. Moreover, powerful tests can mitigate the need of (internal) security pentests. Therefore penetration tests will only have the need to be executed as an external company supervising bought products.

4.5.3 Integration in certification environment. A possible way to increase market adoption is to implement the standardisation in the certification environment of software and companies as follows. Software written that comply to the standard can apply to

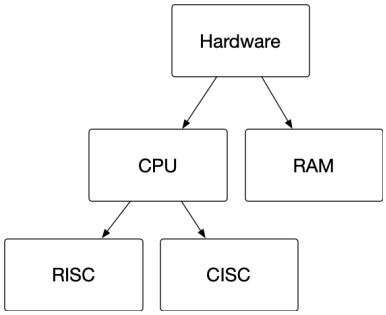

Figure 4: Interface structure for hardware based interfaces. The interface structure is based on the application layer structure without any TCP/IP dependency.

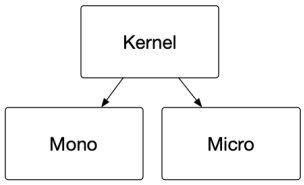

Figure 5: Interface structure for kernel based interfaces. The structure is also based on the application interface without any dependency.

get certified. An issued certificate will then guarantee, that the software is able to understand requests to the standard API of the given versions. A further part of this process is to implement a requirement for companies use at least a specific percentage of software, implementing the standard and therefore possessing the certificate.

5 SUMMARY AND OUTLOOK

The challenge of addressing information security in a heterogeneous system environment has been discussed in the literature from different angles, such as databases [19][22][6], applications [13][21], and system administration [17][18]. Many of these papers clearly point to interfaces being a problem and a solution at the same time.

That is why this paper was aimed at laying the foundation for the introduction of a new, interface oriented perspective, which is building on the analysis described in this paper. At the core of this perspective is an interface oriented view of security design and architectures.

This paper showed that homogeneity is a non achievable goal in IT and therefore any security aspect must include heterogeneous systems. As initial surveys of existing literature and practices [8][28][7] show, this new approach looks promising and can be built to claim international standards such as IEEE, etc. It is expected that many of the current design issues, especially the "piping mess", can thus be avoided by providing this very focused solution. Architecture and design are radically new ways of looking at system environments and might be the best way forward in this situation.

As described in this paper, the next logical step needed towards a solution is the identification of relevant interfaces at the different layer presented in Figure 1 and map them into a naming and parameter scheme as described. This should ideally lead to a pure modular IT system architecture where every component in a layer (hardware, kernel, application) could get swapped with another one respecting the same interfaces. This interface based approach tries to standardise, and therefore ease, inter-process communication. It differs from any other standardisation concepts (e.g.: SQL, POSIX, etc.) from not only concentrating at a specific system, but addressing software architecture in its entirety. Also, it is highly recommended, if not necessary, that this proposed architecture and standartisation is managed and supervised by an independent standardisation organisation (e.g.: ISO, IETF, etc.) for a strong, fast and fair adoption. This solution only addresses the first of the two problem descriptions (non scaling interface landscape; security issues in todays heterogeneity), but may increase security when adding security considerations to the future interface definitions to contribute to the resilience of all components in a product while maintaining full interoperability at the same time. Furthermore automated security tests (e.g.: fuzzing, dynamic code analysis, etc.) will get a lot more flexible and focused on specific problems, when aligning to defined interface standards.

REFERENCES

[1] Florina Almenárez, Andrés Marín, Celeste Campo, and Carlos Garcia. 2004. PTM: A pervasive trust management model for dynamic open environments. , 8 pages.
[2] Joseph T Buck, Soonhoi Ha, Edward A Lee, and David G Messerschmitt. 1994. Ptolemy: A framework for simulating and prototyping heterogeneous systems.
[3] Gang Chen, HV Jagadish, Dawei Jiang, David Maier, Beng Chin Ooi, Kian-Lee Tan, and Wang-Chiew Tan. 2014. Federation in cloud data management: Challenges and opportunities. *IEEE Transactions on Knowledge and Data Engineering* 26, 7 (2014), 1670–1678.
[4] Steven Dawson, Shelly Qian, and Pierangela Samarati. 2000. Providing Security and Interoperation of Heterogeneous Systems. *Distributed Parallel Databases* 8, 1 (2000), 119–145. https://doi.org/10.1023/A:1008787317852
[5] Andreas Eberhart and Stefan Fischer. 2003. *Web Services.* Carl Hanser Verlag.
[6] Peter Fankhauser, Georges Gardarin, M. Lopez, José Manuel Muñoz, and Anthony Tomasic. 1998. Experiences in Federated Databases: From IRO-DB to MIRO-Web. In *VLDB'98, Proceedings of 24rd International Conference on Very Large Data Bases, August 24-27, 1998, New York City, New York, USA*, Ashish Gupta, Oded Shmueli, and Jennifer Widom (Eds.). Morgan Kaufmann, 655–658. http://www.vldb.org/conf/1998/p655.pdf
[7] Scott R. Gallagher. 2012. The battle of the blue laser DVDs: The significance of corporate strategy in standards battles. *Technovation* 32, 2 (2012), 90–98. https://doi.org/10.1016/j.technovation.2011.10.004
[8] Robert Heininger, Loina Prifti, Markus Böhm, and Helmut Krcmar. 2016. Towards a Model of Heterogeneity in IT Service Value Networks: Results from a Literature Review. In *29th Bled eConference: Digital Economy, Bled, Slovenia, June 19-22, 2016*. AIS, Bled, 37. http://aisel.aisnet.org/bled2016/37
[9] AO Hirshman. 1964. The paternity of an index.
[10] Alexis P Jacquemin and Charles H Berry. 1979. Entropy measure of diversification and corporate growth. *The journal of industrial economics* (1979), 359–369.
[11] Yong Jin Kim and Kichan Nam. 2009. Service Systems and Service Innovation: Toward the Theory of Service Systems. In *Proceedings of the 15th Americas Conference on Information Systems, AMCIS 2009, San Francisco, California, USA, August 6-9, 2009*, Robert C. Nickerson and Ramesh Sharda (Eds.). Association for Information Systems, San Francisco, 1. http://aisel.aisnet.org/amcis2009/1
[12] Paul Kocher, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, and Yuval Yarom. 2018. Spectre Attacks: Exploiting Speculative Execution. *CoRR* abs/1801.01203 (2018). arXiv:1801.01203 http://arxiv.org/abs/1801.01203
[13] Lucio La Cava, Sergio Greco, and Andrea Tagarelli. 2021. Understanding the growth of the Fediverse through the lens of Mastodon. *Applied Network Science* 6, 1 (2021), 1–35.
[14] Paul Lackner. 2021. *Security thoughts on modern software development.* Master's thesis. UAS St. Pölten.

[15] Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, and Mike Hamburg. 2018. Meltdown. *CoRR* abs/1801.01207 (2018). arXiv:1801.01207 http://arxiv.org/abs/1801.01207

[16] Lloyd's and Cyence. 2017. Emerging Risks Report 2017 - Technology, Counting the cost - Cyber exposure decoded. https://assets.lloyds.com/assets/pdf-emerging-risk-report-2017-counting-the-cost/1/pdf-emerging-risk-report-2017-counting-the-cost.pdf

[17] Aymeric Mansoux and Roel Roscam Abbing. 2020. *SEVEN THESES ON THE FEDIVERSE AND THE BECOMING.* Institute of Network Cultures, Amsterdam, and transmediale e.V., Berlin.

[18] Félix Gómez Mármol and Gregorio Martínez Pérez. 2010. Towards pre-standardization of trust and reputation models for distributed and heterogeneous systems. *Computer Standards & Interfaces* 32, 4 (2010), 185–196.

[19] Sylvia Melzer, Hagen Peukert, Hongxu Wang, and Stefan Thiemann. 2022. Model-based Development of a Federated Database Infrastructure to support the Usability of Cross-Domain Information Systems. In *2022 IEEE International Systems Conference (SysCon)*. 1–8. https://doi.org/10.1109/SysCon53536.2022.9773811

[20] Justice Opara-Martins, Reza Sahandi, and Feng Tian. 2014. Critical review of vendor lock-in and its impact on adoption of cloud computing. In *International Conference on Information Society (i-Society 2014)*. IEEE, London, 92–97.

[21] Aravindh Raman, Sagar Joglekar, Emiliano De Cristofaro, Nishanth Sastry, and Gareth Tyson. 2019. Challenges in the Decentralised Web: The Mastodon Case. *CoRR* abs/1909.05801 (2019). arXiv:1909.05801 http://arxiv.org/abs/1909.05801

[22] Tomas Skripcak, Claus Belka, Walter Bosch, Carsten Brink, Thomas Brunner, Volker Budach, Daniel Büttner, Jürgen Debus, Andre Dekker, Cai Grau, Sarah Gulliford, Coen Hurkmans, Uwe Just, Mechthild Krause, Philippe Lambin, Johannes A. Langendijk, Rolf Lewensohn, Armin Lühr, Philippe Maingon, Michele Masucci, Maximilian Niyazi, Philip Poortmans, Monique Simon, Heinz Schmidberger, Emiliano Spezi, Martin Stuschke, Vincenzo Valentini, Marcel Verheij, Gillian Whitfield, Björn Zackrisson, Daniel Zips, and Michael Baumann. 2014. Creating a data exchange strategy for radiotherapy research: Towards federated databases and anonymised public datasets. *Radiotherapy and Oncology* 113, 3 (2014), 303–309. https://doi.org/10.1016/j.radonc.2014.10.001

[23] K. Spichale. 2019. *API-Design: Praxishandbuch für Java- und Webservice-Entwickler.* Dpunkt.Verlag GmbH.

[24] Ayman Tajeddine, Ayman I. Kayssi, Ali Chehab, and Hassan Artail. 2006. PATROL-F - A Comprehensive Reputation-Based Trust Model with Fuzzy Subsystems. In *Autonomic and Trusted Computing, Third International Conference, ATC 2006, Wuhan, China, September 3-6, 2006, Proceedings (Lecture Notes in Computer Science, Vol. 4158)*, Laurence Tianruo Yang, Hai Jin, Jianhua Ma, and Theo Ungerer (Eds.). Springer, Wuhan, 205–216. https://doi.org/10.1007/11839569_20

[25] Edward Tenner. 2015. The design of everyday things by Donald Norman. *Technology and Culture* 56, 3 (2015), 785–787.

[26] Inna Vistbakka and Elena Troubitsyna. 2020. Analysing Privacy-Preserving Constraints in Microservices Architecture. In *44th IEEE Annual Computers, Software, and Applications Conference, COMPSAC 2020, Madrid, Spain, July 13-17, 2020.* IEEE, Madrid, 1089–1090. https://doi.org/10.1109/COMPSAC48688.2020.0-126

[27] Inna Vistbakka and Elena Troubitsyna. 2020. Formalising Privacy-Preserving Constraints in Microservices Architecture. In *Formal Methods and Software Engineering - 22nd International Conference on Formal Engineering Methods, ICFEM 2020, Singapore, Singapore, March 1-3, 2021, Proceedings (Lecture Notes in Computer Science, Vol. 12531)*, Shang-Wei Lin, Zhe Hou, and Brendan P. Mahony (Eds.). Springer, 308–317. https://doi.org/10.1007/978-3-030-63406-3_19

[28] Thomas Widjaja, Jasmin Kaiser, Dennis Tepel, and Peter Buxmann. 2012. Heterogeneity in IT Landscapes and Monopoly Power of Firms: A Model to Quantify Heterogeneity. In *Proceedings of the International Conference on Information Systems, ICIS 2012, Orlando, Florida, USA, December 16-19, 2012.* Association for Information Systems, Orlando. http://aisel.aisnet.org/icis2012/proceedings/BreakthroughIdeas/3