Student Number: 150151476
Student: Christopher Uwe KLOOZ

Because Linux is law: mapping the socio-technical architecture around Linux and determining its institutional dynamics

Submitted as part of the requirements for the award of the
MSc in Information Security
at Royal Holloway University of London.

Information Security Group
Royal Holloway, University of London
2022-23

Table of Contents

Abstract

International Relations, Information Security and other fields with security considerations take much for granted: however, traditional assumptions are based on social systems and environments, but in today's socio-technical systems/environments, much has changed. Indeed, the security foundations and provisions at the most fundamental level, in anarchy, where we want to establish order through "sovereigns" (today, sovereigns are mostly assumed to be sovereign states with their given architectures), has changed: the Internet itself has become a new anarchy that needs order to achieve security, and any organization/society/state that has become dependent on the Internet directly or indirectly, exists in socio-technical anarchy, no longer just social/physical anarchy.

However, the "working", "running" Internet is taken for granted, while current thoughts of regulation and of achieving security only consider what happens on top of the Internet: yet, Linux is the kernel that drives the majority of the infrastructure of the Internet and on the Internet. Current regulation and security approaches that aim to achieve security take for granted what Linux operating systems and Linux firmwares provide. Further, much that has become a common demand and on which societies and states of today already depend on has risen in Linux (even states have pressures to adopt, e.g., Cloud technologies, which are based mostly on Linux technologies such as KVM): the new socio-technical anarchy consists of more than the known social/physical anarchy that we have to take as it is. Instead, it also consists of a new type of anarchy that is actively and intentionally developed by people, and the outcome is the "working", "running" Internet with its agency, which leads to notions of "security THROUGH anarchy". Indeed, even technical protocols and technologies depend on what the kernel provides - and that it is provided stably and securely and on time (which includes implementations of regulations, protocols or algorithms). In such respects, organizations like the "IETF" or the "NIST" illustrate overlaps of the two systems and their architectures.

People and organizations that develop the Linux kernel contribute to global socio-technical security provision for anything and anyone who/that depends directly or indirectly on the Internet (which makes its architecture a sovereign in some perspectives): the Linux architecture actively and intentionally develops and maintains some of the anarchy (which is "Linux" driven) in which further regulation and security means can be put in place, and both can be put in place only if/when supported properly by the kernel, while the latter already develops the anarchy of tomorrow, which shapes the regulations/means of the day after tomorrow (such as Cloud technologies). Yet, this does not replace the existing anarchy in which sovereign states remain important security providers, which also influence with today's regulations what is implemented in the kernel, but assumptions that the traditional system of sovereign states and their architectures are omnipresently dominant introduces fatal presumptions for socio-technical security provisions.

However, while the majority of organizations and societies still struggle to achieve security when it comes to socio-technical environments, Linux remains capable to manage one of the most complex pieces of technology in history: it remains capable to persistently fulfill and tailor to demand, while it remains capable to provide sufficient security and stability on the most complex levels (both in the development stages but also in the resulting product). Indeed, solving security issues "on time" in these realms can be often a question of hours. The socio-technical Linux architecture is much more than just an organization that develops a technical operating system kernel. The architecture that develops and maintains the Linux kernel completely differs to other types of architectures. Indeed, it can be questioned how far it can be compared to existing thoughts of architectures, but also how far and when it can be distinguished from other architectures, and when there are overlaps or blurs.

I developed new approaches that avoid biases and prejudices: I "follow" problems during their processing in the Linux architecture and "follow" how/where these problems are solved from a socio-technical perspective, and I consider and analyze what appears on the way of problem solving, avoiding predetermined presumptions. During the "following", I focused on the institutions of Linux, which facilitate and foster solving problems. Determined by institutions, I derived "conflict actors" comprehensibly and reproducible. Indeed, determining the actual actors that are interacting and engaged in problem solving, and identify how and where they are distinguished from each other but also how they develop, is a critical task, whereas working with pre-determined and static actors can be itself misleading for research.

Further, I keep using the existing "secure by design" approach to evaluate socio-technical systems in a holistic way for their security provision: avoid single points of failure, and ensure intended input. Indeed, the architectures in the traditional system in anarchy often fail to meet these properties when they are exposed to socio-technical anarchy, where information technology can break the traditional separation of powers and thus create single points of failure, which might get fatal if complemented by inputs that are not intended by this traditional system.

Generally, the Linux architecture, which provides security at (and in) levels of anarchy next to other entities such as states, deploys no command-economy-based "separation of powers" but a competition-based "distribution of powers". At the same time, its institutional dynamics achieve that competition is among institutions and not among people, while people are major determinants of the organizational structures and boundaries but also of its overlaps. The stable balance of competition and collaboration has correlations to the research of AnnaLee Saxenian with regards to the Silicon Valley-related economies. Generally, much of the security provision and problem solving capabilities in the Linux architecture is shaped by the competition(s) of and in its institutional dynamics and in its environment.

Further, the institutions and their dynamics (and how they shape/determine/distinguish actors and behaviors) are not just strongly different to other architectures (including many different institutions that contribute to socio-technical security and stability but that have not yet been related to security considerations), they also achieve that known institutions/behaviors create different agency in the Linux architecture than in other architectures: indeed, a basic human behavior that is seen as major security risk in traditional thoughts about organizations/states behaves in the Linux architecture as a means to increase security.

Another difference is that security is no dedicated consideration: it is only about problem solving. Indeed, everything is related to some type of security, and dedicating security can thus be argued to be generally artificial. However, when it comes to privacy, there is an exception, a positive correlation: security dedication where privacy is needed, and no dedication where privacy is not necessary. Yet, the privacy needs of this architecture are much lower than in other architectures, while it has to be kept in mind that privacy needs of societies can also change formally and informally.

In the end, it can be argued that there are two overlapping but strongly different systems (so much differently constituted that they cannot be clearly compared with or distinguished from each other) that provide security foundations and provisions in and on top of anarchy. And it cannot be excluded that societies and organizations that suffer from security issues do so because they tailor to the wrong system "in" (or "of") anarchy, and thus adopting the wrong institutions, when they are exposed to the socio-technical environment of today (and its socio-technical anarchy). However, there are many open questions that future research has to answer, and vulnerabilities of the Linux system cannot be excluded. Furthermore, this project uncovered only small amounts of this architecture.

Because Linux is law: mapping the socio-technical architecture around Linux and determining its institutional dynamics

1. Introduction, context & socio-technical systems

1.1. Today's socio-technical societies depend on the Internet and thus on its security

Modern societies are no longer just social but have become increasingly socio-technical: this means that the points of transition between cyber and real world realms are blurred and omnipresent today (Klooz, 2021, p.4,14,63). It is no longer possible to clearly distinguish between the cyber and the real world: there are permanently transnational, "trans-world", reciprocal interactions and impacts, with both worlds shaping each other (ibid, p.14,63). When societies, cultures and their organizations have begun to align and exchange globally through the Internet while additionally new Internet-centered online communities have appeared, all of them started to rely on the Internet.

Online communities can be highly productive value creators on whose "value output" other societies, cultures and their organizations can depend on while the online communities remain dependent on the other because a "pure online existence" remains impossible. Much of these "value outputs" are technologies that are taken for granted by many today, while these technologies keep maintained online by one or many entities - like communities - and their interactions. Often, these entities are intangible in the real world.

Related technologies might include the Linux kernel that drives the majority of servers on the Internet and also much of the infrastructure of the Internet itself (W3Techs, 2023a; 2023b; IDC, 2020; Juniper, 2021; Cisco, 2021a; 2021b), which will be more deeply elaborated later. Other related technologies include the web browsers we use as of today and especially their "engines" that make up their capabilities, both for features and security: web browsers and web engines are the outcome of strong competition with information flows and knowledge aggregation among many interrelated and even overlapping organizations and communities (e.g., Simmons, 2022; Mozilla, 2022a; 2022b; Foley, 2019; Microsoft, 2018; 2022): today, both people and organizations depend on such technologies and their proper maintenance.

The advantages of having the Internet embedded in modern societies have proven to be so competitive that separating these realms is no longer realistic. Indeed, cyber attacks on governments brought new means into the International Society of mutually-recognizing & UN-charter-constituting sovereign states (hereinafter "International Society"). Cyber attacks challenge established institutions like (but not limited to) the presumption of innocence through the absence of legally-sufficient evidence (Klooz, 2021, p.9,16,26,27,28): however, none of the developed sovereign states have yet started to consider to go "offline" again. "offline" tendencies can also not be observed among their citizens and industries. Today, achieving competitive advantage in a field is no longer realistic without making use of the Internet, directly or at least indirectly: this applies to private and public entities, and to individuals, small- and mid-sized but also to big multinational entities.

Consequently, "social contracts" through which people pass some of their freedom to a common sovereign state in order to achieve types of security (Castiglione, 2015) have become "socio-technical contracts" (Klooz, 2021, p.11,63,65): considerations and research around the provision of security in modern societies have to integrate the security of and on Internet.

1.2. The Internet is different(ly governed), it depends on the Linux kernel, and both remains neglected

Unlike social systems, the technical system(s) of the Internet has to be intentionally and actively developed and maintained by people. This technical foundation determines what is possible and what can happen on and with the Internet. However, social governance (in social systems instead of socio-technical systems) in societies is not used to consider such intentional, active development and maintenance of its environment: with regards to International Relations theory, the dominant governance in social systems is to create order in anarchy, whereas anarchy is the lack of a ubiquitous superordinated authority and thus, the lack of a sovereign who creates order in anarchy (Milner, 1991).

However, social system governance and related science disciplines are not used to actively and intentionally develop and maintain the conditions of the underlying anarchy, in which order has to be established. Indeed, the need and capability to develop the underlying anarchy leads to notions of "security THROUGH the anarchy (which we develop/maintain)" instead of "security IN anarchy". All this questions contemporary notions of "anarchy" (ibid), "sovereignty" (Philpott, 1995; Chinkin & Kaldor, 2017, pp.85-86) and any approach or theory that takes them for granted. Although adjusted notions such as in Chinkin & Kaldor (2017, pp.38-86) are in debate, they are not linked to holistic socio-technical perspectives, whereas I could not identify research with socio-technical considerations that involves related security provisions as considered in my project: socio-technical approaches do not yet seem to be used to analyze the foundations on which all other security provisions are depending and relying on, and by which they are shaped. At the same time, it can be questioned which actors of today fulfill the currently established notions of a sovereign (and in which respect), and who is thus involved as a provider in the socio-technical contracts.

Therefore, current approaches around creating and integrating Internet governance (both in practice and research), and establishing and enforcing regulations on the Internet and therefore, creating security of and on the Internet, keep focused on the outcome of the development and maintenance of the Internet: the "working", "running" Internet is taken for granted. Examples that take it for granted include Margetts, et al. (2021): "The Internet and public policy: Future directions", which consolidates related efforts of the *Policy & Internet* journal in a "collection of essays" that "highlights the past ten years of the journal and paints a clear trajectory for the next era of Policy & Internet" (ibid.). The fundamental foundations of today's security capabilities and provisions are neglected. Correspondingly, current approaches and research are likely to oversee actors and institutions that already actively but maybe also intuitively establish security in the realms that are taken for granted by others, whereas the related actors maybe do not perceive themselves as involved in the socio-technical contracts. Again, Margetts, et al. (2021) take traditional institutions and traditional sovereigns (and who the sovereigns are) as well as their roles in solving security challenges and in providing security for granted. Another example is Roberts, et al. (2011).

At the same time, security research that considers social or psychological aspects in conjunction with technical aspects (may it be a holistic socio-technical or another approach) tends to focus on organizations and takes not just their immediate environment for granted, but the whole system in anarchy. This leads such research to neglect the impacts of and incentives from an environment on the behaviors and cultures that occur and develop within: behaviors and cultures (and the institutions that make these behaviors/cultures) are not only the outcome of interactions and interrelations among each other, but also the outcome of the incentives/impacts of/from their environment within which they interact and interrelate. Indeed, human behavior (and thus the behavior of organizations that are shaped by humans) has to be considered and understood in conjunction with its environment, and not considered on its own with the environment being seen as static constant that is taken for granted, as done by Fogg (2009).

Additionally, this makes such security research also to oversee actors/institutions/behaviors/cultures and developments that are excluded by its unproven (or at least, no longer proven) presumptions: actors/institutions/behaviors/cultures such as those that ensure the "running", "working" Internet remain underresearched and not considered, despite their immediate capabilities in combining satisfying needs/demand with enabling and providing types of security (and foundations for security measures/means that can be deployed/exploited within this system) up to the level of anarchy. Such security research tends to assume that the surrounding system remains the traditional one, which can be itself a

corrupting factor: indeed, ignoring potential impacts and interrelations with the system that provides the "working", "running" Internet can lead research outcomes to be false due to false presumptions, risk that developing variables are presumed to be constants throughout subsequent research projects, and avoid to learn from and adjust to (and thus exploit the possibilities provided by a) system that seems to ensure high levels of flexibility, efficiency and security.

Indeed, the "running", "working" Internet and its actors/institutions/behaviors/cultures seem to provide sufficient security to make their impact not apparent because it sufficiently "works": with regards to the research of security cultures, university programmes consider research of security cultures such as Alshaikh (2020) that considers different organizations, or Georgiadou et al. (2020) that helps to evaluate and analyze cultures within organizations, but both assume the environment/system to be a given constant. However, what if some (social, psychological, technical) security problems are themselves only the result of organizations that tailor and adjust to the traditional system (and its institutions) while another (completely different) system has already taken over the security provision/foundation in the given field/realms? This question links not only to a "system" or "actors", but to a whole architecture with its institutional framework, which could even co-exist to the traditional system and overlap with it, partly or fully in anarchy. Indeed, this is a major question that underlies both my 2021 thesis and also this project.

As I have already elaborated in my 2021 thesis (Klooz, 2021, p.19,58,59), a large number of servers on the Internet are running operating systems that are based on the Linux kernel: it is the most widespread kernel on Internet servers with 38.75-80.66% in 2023 (W3Techs, 2023a; 2023b) (0.85-3.76 % increase of Linux-driven servers compared to my 2021 thesis), whereas the statistics of the other Unix-like operating systems (W3Techs, 2023a) indicate that the majority of the unknown Unix servers are Linux-driven as well, which means that the actual Linux use is likely to be closer to the 80.66% than to the 38.75%. At the same time, Android is the dominant operating system of the majority of smartphones and tablets: 71.47%-71.59% in 2021 (Statcounter, 2023). Each Android kernel keeps derived from and dependent on the Linux kernel, more precisely from one of Linux's Long-Term Support *branches* that keep receiving updates from the Linux community for a long period (Android Open Source Project, 2022a; 2022b; 2022c; 2022d).

However, the above statistics focus on what happens on the Internet, but are not evident for the Internet's own infrastructure itself: when analyzing it in my 2021 thesis, Juniper and Cisco together had a market share of 43.7% in 2020 in the market of carrier-grade hardware (Klooz, 2021 with IDC, 2020), which makes up the dominant layer of the Internet's infrastructure to which all subordinated layers have to tailor to and on which they depend on (also in terms of security). When analyzing for my 2021 thesis, Cisco and Juniper had already decided and fixed that all their non-Linux-based operating systems (or, firmwares) were to be replaced by Linux-based replacements (Klooz, 2021 with Juniper, 2021; Cisco, 2021a; 2021b). For Huawei, which had in 2020 the second biggest market share after Cisco but before Juniper (ibid), there was no binding information about how far they deploy Linux on carrier-grade hardware or not. Therefore, the Linux relevance could have been already much higher back then.

However, today, without considering still supported legacy software that has not yet reached its end of life, Cisco deploys Linux on its three firmwares for its carrier-grade hardware (Cisco, 2023a; 2023b; 2023c), and Arista Networks (which has taken over Juniper's market position: IDC, 2020; 2022) also deploys Linux on its firmware for its carrier-grade hardware-related (Arista, 2023), while both companies do not offer non-Linux alternatives for this market: in the 4th quarter of 2022, these two companies had together a market share of 54.9% (IDC, 2022), with more Linux-driven market shares being possible depending on the other companies in this market. Therefore, the infrastructures of the Internet (primarily carriers) and on the Internet (servers on the Internet; non-carrier service providers) but also many clients (such as Android) depend to a large extent on what the Linux kernel provides and supports.

Once the Linux kernel provides something, it will enter a global socio-technical and transnational competition and if it proves competitive, it will be deployed in the socio-technical society and maybe even create pressures to use it (which can develop to dependency): e.g., generally resisting the use of Cloud computing can make many businesses less competitive, or even threaten their competitive advantage at all. Indeed, Cloud computing is in most cases Linux-based and Linux-dependent as I will elaborate in the coming paragraphs (Sharwood, 2017; Amazon, 2022; Vaughan-Nichols, 2018; Levin, 2019; Google, 2022; Statista, 2022).

However, this does imply that there are no other types of Internet-related competition: e.g., Linux may implement standards that have been standardized by the Internet Engineering Task Force (IETF) or the National Institute of Standards and Technology (NIST). Yet, since standards have themselves no agency, they (and the competition of and around them) remain dependent practically on Linux: it cannot be said that the competition that made the NIST's Speck ciphers disappear was Linux-related. But it can be said that since the Speck ciphers were removed officially from the Linux kernel any realistic possibility for Speck's adoption even for any use cases has been finally buried. On the other hand, adding the stream cipher ChaCha20 to the kernel in 2015 (documented in the Linux kernel source code repositories, which are elaborated later) is next to Google's adoption of ChaCha20 for mobile cryptography (Google, 2015) likely to be the major trajectory that led to ChaCha's rise to today's dominant stream cipher that has replaced even AES in several use cases.

There are many ways and competitions that can end up and lead to developments in the Linux kernel but also in alternatives that are still existent in several fields. But putting something to the wider socio-technical anarchy of the Internet to finally prove competitive in the global socio-technical system of socio-technical societies is today highly dependent on and impacted and filtered by Linux. Even if an alternative puts forward something that ends up in the global system, it is likely to have to pass and succeed in the Linux kernel before wider adoption, spread and impact becomes likely. Practically, the Linux kernel is a major determinant of the agency of and on the Internet.

The scalability, efficiency and price/performance ratio of Linux-host-based Cloud services available from providers like Amazon, Microsoft and Google (Sharwood, 2017; Amazon, 2022; Vaughan-Nichols, 2018; Levin, 2019; Powell, 2019; Google, 2022) have already proven competitive: the Cloud infrastructures of Amazon, Microsoft and Google are Linux-based (ibid) and have a consolidated Cloud market share of 66% (Statista, 2022). If their remaining competitors follow the same pattern, it is likely to be much more Linux in this market. Furthermore, the Linux dominance also applies to the virtual instances running on these Cloud infrastructures: even on Microsoft's Azure Cloud, Linux instances have surpassed Microsoft Windows instances at the latest in 2019 (Levin, 2019).

On one hand, many related technologies such as the Kernel-based Virtual Machine "KVM" (which is a core technology at least in Google's and Amazon's current Cloud solutions (Amazon, 2022; Google, 2022)) have been introduced to the socio-technical societies by Linux, making Linux a major determinant in what becomes possible and thus enters competition towards adoption in societies and therefore, impacts them. On the other hand, Linux remains the most competitive and efficient solution to provide these technologies and also determines which prices (and thus use cases) are finally realistic. The dominance of Linux on Internet servers makes it also one (if not the major) determinant in terms of the security of and around the related services (and everything that depends on them): the kernel rules, controls and governs everything on the system. The kernel can break and impose everything on the system the kernel supports and vice versa, what the kernel does not support directly or indirectly is not supported by the system. Correspondingly, governing the kernel implies governing the system. The power and role of a kernel is not related to Linux but a general rule on modern operating systems.

With Wireguard, which has been introduced with the Linux kernel 5.6 (kernel.org, 2020a), there is another technology that has started to spread, being adopted to other operating systems as well and adopted by several service providers within their own infrastructures but also provided to their customers. Indeed, its performance, simplicity, security characteristics and its flexibility (flexibility not in terms of algorithms but in terms of being adaptable to different types of solutions, e.g., because it does not impose any restrictions on the use of authentication mechanisms) outperform existing standards like IPSec or TLS and related solutions in several use cases (Donenfeld, 2017; Donenfeld & Milner, 2017; Lipp, Blanchet & Bhargavan, 2019). If its increasing adoption will lead it to be contained in future standards remains to be seen.

However, given the large use of Linux on servers, existing standards like TLS or IPSec are bound to the Linux kernel adopting and maintaining (or at least enabling) it: a formal standard of the IETF is not necessarily to become an informal standard. Indeed, if organizations like the IETF do not adapt to the demand, the demand will will bypass them (and create pressures to later adopt the de facto standards that have emerged in competition): we have already seen

this, e.g., in the de facto standardization of Curve25519 in the de facto (and most widespread) reference implementation of OpenPGP (GnuPG Project, 2014; Callas, et al., 2007; Jivsov, 2012), followed by increasing related deployments of Curve25519. Today, Curve25519 has developed to one of the dominant elliptic curves in cryptography.

However, organizations like the NIST and the IETF have themselves also the power to create incentives and impact supply and demand: what these organizations standardize has an increased likelihood to become a demand that often has to be satisfied by the kernel, leading it to become supplied by the kernel. In any case, unconditional unilateral dominance should not be assumed but dynamics, interrelations and interactions. Also, it might be noted that these organizations and the Linux kernel community are likely to have overlapping members and tend in most circumstances to be complementary to each other, acting in collaboration. Indeed, the selection processes that culminated in the NIST's Advanced Encryption Standard or the Secure Hash Algorithms 2 and 3 or the NIST-unrelated Argon2 indicate comparable institutions to develop solutions, and the Linux kernel is a common next stage after algorithms have been standardized, with the kernel stage and the more theoretical preceding stages establishing feedback loops.

In any case, Wireguard's use cases and increasing adoptions make it increasingly relevant for the security of confidentiality, integrity and availability (CIA): technologies on which we rely on are generally linked to the security of availability. Without reliable/adaptable and efficient but also cost-effective encryption, the transfer of critical data would not be possible, which links to the security of confidentiality and integrity. Without the transfer of critical data, many use cases would be not possible, which again links to the security of availability (of some services). With this in mind, we also depend on the kernel's implementation of, e.g., IPSec, and additionally, the kernel's capabilities to control and protect related data and processes (e.g., mandatory access controls). Complementary, if technologies are adopted by the kernel, this automatically leads these technologies to be distributed and deployed to many systems globally.

Therefore, the development and maintenance of such technologies within the Linux kernel is linked to the security of CIA in general. With all the above in mind, and when set in contrast to Lawrence Lessig's assumption around "code is law" (Lessig, 2006), it can be derived: Linux is law. Furthermore, the technologies and use cases that have been triggered by Linux and that have later been adopted in solutions or ported to other kernels (e.g., efficient kernel-based KVM virtualization, Wireguard, Kubernetes), and integrated directly or indirectly into modern societies (e.g., KVM virtualization, Xen, Kubernetes, Cloud infrastructure-related technologies) indicate "imitation" towards Linux and its agency: with regards to the International Relations perspective: imitation in anarchy can be argued to be one of the major indicators of power or even dominance in a given realm.

1.3. What research around the provision of security keeps neglecting

In my 2021 thesis (Klooz, 2021, pp.10-11,pp.15-16,p.35) I already elaborated how research around security keeps focused on assumptions and institutions from social environments without questioning them in the new socio-technical environment. Indeed, I could not identify debates that evaluate traditional institutions like the presumption of innocence or the traditional separation between the judicial, executive and legislative powers. Instead, related research institutions like Harvard's "Berkman Klein Centre for Internet & Society", Hertie School's "Centre for Digital Governance" & its "Centre for International Security", or the "Oxford Internet Institute" took traditional institutions from social environments for granted without questioning their security capabilities in socio-technical environments (ibid,p.10). The collection of Margetts, et al. (2021) remains an example that is related to the "Oxford Internet Institute" but also to other research centers. Another example is Roberts, et al. (2011) from the "Berkman Center for Internet & Society".

This again illustrates that social system era notions and assumptions remain unchallenged in the socio-technical era: everything up to and including the running Internet is taken for granted. Indeed, at the scientifically related above mentioned research institutions' projects, I could neither identify considerations of Internet-related software development and maintenance with their institutions, nor general considerations of the intentionally and actively developed socio-technical anarchy of the Internet.

There are research projects that go deeper and consider institutions like the IETF, such as the "Mapping the Operational Institutional Complex Sustaining the Internet's Infrastructure" project (Sowell, 2022) from the University College London's Department of Science, Technology, Engineering and Public Policy, and related earlier publications (Sowell, 2012; 2019; Brass & Sowell, 2021).

However, standards that are standardized through institutions like the IETF have to be implemented and maintained in the form of code to get agency and to enter the global socio-technical system. Additionally, this code has to run on an operating system. The latter has to provide a secure, stable and resilient but also cost-efficient environment: without that, the best Transport Layer Security (TLS) implementation is not worth much.

So considering institutions like the IETF still neglects the security-dependence on high-quality and transparent code implementation and maintenance (of both the standard and its environment), independent and distributed review and testing, fast identification and solving of bugs and security flaws (including the fast forwarding of critical information to and feedback loops with responsible entities), providing functions like firewalls and mandatory access controls, and much more: everything that makes the Internet running, and keeps it running sufficiently securely and stably, and all this needs to be done in a sufficiently efficient and flexible way.

Furthermore, the considered standardizations around the Internet (e.g., IETF, NIST) are focused on the generic and common needs: transport & application layer security protocols, cryptographic algorithms, internet protocols (IPv4, IPv6), standardized data transfer (HTTP(S)), and so on.

Yet, this does not standardize the possibilities and interrelations (and of course implementations) that rise from protocols like HTTP(S), which includes risks that rise from them, especially if these possibilities prove competitive and thus become integrated into socio-technical societies, which at some point depend on it. Indeed, neither social nor socio-technical societies wait for standardization, and not everything can be standardized, for many reasons.

Today, much is realized through HTTP(S), including different types of streaming, conferencing, system updates, system or cloud administration, enterprise resource planning (ERP) and much more. The HTTP(S) standards themselves cover only a little part of what can cause problems for the security of CIA.

Additionally, information about demand and the demand itself, which later develops to standards from organizations like the IETF, has to come from somewhere. It is the same for the standard proposals which enter competition towards standardization. Even in cases where the IETF develops a standard as intended, there is much more than just the formal entity we see before us. On the other hand, the previously mentioned example of Curve25519 has already indicated that organizations like the IETF should not be automatically assumed to "make" the related competition but that they possibly are only moderators of competition, being bypassed when not considering the competition's incentives.

Organizations like the IETF can provide incentives through standardization to organize and harmonize generic frameworks of protocols but also to provide a common place to evaluate different proposals. However, we should always keep in mind that web browsing and related services did not start with the official standardization of HTTP. Standardization only unified the provision of these services and facilitated common ground for the different browsers when the demand was already evident. Organizations like the IETF are neither the start nor the end. Also, they cover only a small part of the Internet-related security-relevant issues, and depend on and are forced to respond to what code provides. And Linux is the basis for much of the code that makes the Internet: Linux accommodates, organizes, harmonizes and practically standardizes the code foundation of the Internet, forming at least often (maybe mostly) the beginning and the end of the related competition and provision. However, notions with "beginnings" and "ends" might be misleading at all and should be replaced by loops, more precisely bidirectional feedback loops. Yet, the Linux institutions are maybe not different to, some maybe even the same than, the institutions of the IETF: the way of standardization at the IETF might be compared to the way code is added to the Linux kernel.

Developers and companies who want their code to be part of Linux, have to compete with others towards successful *merge requests* accepted by the Linux kernel community. And the *merge requests* might also impact future standards. In "socio-technical anarchy", Linux provides order where other sovereigns yet remain silent - or from an alternative perspective, it provides "ordered anarchy". Obviously, this does not mean that Linux fulfills the provision alone, and its feedback loops are not a closed system.

However, that Linux (and likely others) remain(s) hidden to much (if not most) holistic security research could indicate that its socio-technical architecture properly fulfills its role silently, whereas we do a lot of research in other areas where security problems are apparent. These apparent security problems could be linked to misunderstandings or false assumptions related to the silent role of, e.g., Linux, and even if not, explicit research about their silent role can still provide knowledge to tackle the apparent security problems more effectively.

Additionally, Internet-related organizations that have a formal role remain taken for granted once their role is enshrined in statues or law, without questioning if they fulfill their role in practice, to which extent they fulfill that role and if there is more to be fulfilled or to be considered. Vice versa, organizations and institutions that have no formal role or no formal body remain widely neglected and thus, their roles and the demand they meet not clearly evaluated. Obviously, this also keeps their institutions hidden, and how far these institutions could solve problems also elsewhere.

Also widely neglected is the wider architecture, to which both Linux and the IETF (and others) belong to, that is based on a competition-based "distribution of powers" (it is ensured that everything is publicly available/discussable, that it can be done/conducted by every capable entity without pre-determination or fixed allocation, and it is always the entity chosen that fits the situational demand and the "will" of the related community best, but it can and will be replaced by other entities once it abuses power or once it is no longer is competitive for other reasons) instead of a command economy-based separation of power, which I identified in my 2021 thesis (Klooz, 2021, p.44,64): some of the related institutions are legally and technically transparent design and development, public review, distributed testing, and forking. These institutions create much security in the wider architecture that surrounds and contains Linux, among others to avoid monopolies that could be abused, and to make technology resilient to attacks (ibid, p.4,46,54,64). Since these institutions are clearly incorporated in Linux's socio-technical architecture (ibid, pp.20-21,pp.38-39,p. 42,55,58), they are likely to become also relevant in this project.

2. An institution-based approach to identify how the socio-technical Linux architecture solves problems

2.1. Approach & concepts

In my project, I will analyze and map the security-relevant socio-technical architecture around the Linux kernel through the problems it solves: solving problems is the foundation for managing any type of (direct and indirect) security issue and thus, for providing security. On one hand, the technology/means (e.g., code, standards) that give rise to security issues are themselves the solution of problems. On the other hand, the sufficiently-quick and reliable solving of security issues is itself problem solving. With regards to supply and demand, it can be argued that "demand" is at its foundation the need to have one or many problems solved. "Supply" on the other hand contains solution candidates, and sometimes, supply makes the demand aware of problems that had not been known explicitly before.

This project is about problem solving in socio-technical environments where the boundaries between social and technical elements are blurred and fluent, without distinguishable points of transition, and where impacts, interrelations and interactions that are boundary-less are omnipresent (Klooz, 2021, p.4,14,63). Therefore, any differentiation between social institutions and technical functions (or, subroutines) could be misleading, and for the focus of this project irrelevant anyway: finally, technical functions and social institutions describe the same but for environments that used to be treated separately.

In my 2021 thesis, I already elaborated the ease of considering social institutions and technical functions in a "merged" way for the research of socio-technical systems:

> *"On the lowest level, software is shaped by data and behavior. Subroutines (e.g. "functions") incorporate behavior, possibly inherent data and they can be "called" to get activated. The call can contain further data that gets processed and then output by the subroutine. The sum of all calls makes the agency of the caller. On the highest technical level, the application itself is the caller. Also, subroutines can call each other."*
>
> *"Just like institutions, subroutines are "stable, valued, recurring patterns of behavior" (Huntington,1968,p.12) and a "behavioral manifestation" (ibid.,p.10), although the manifesting consensus is mathematical, not moral. Such correlations between technical and social open systems enable the transfer of approaches."*
>
> — *Klooz 2021 p.13*

In short, institutions are for people what functions are for machines. Therefore, in this project, I will not differentiate between institutions and functions/subroutines but refer to both as "institutions".

Another concept that will be deployed in this project's socio-technical research is "secure by design" from Santos, Tarrit and Mirakhorli (2017): like the merging of institutions and functions/subroutines in socio-technical research, I already elaborated the possibility to deploy the "secure by design" software engineering approach to socio-technical research in my 2021 thesis (Klooz, 2021, pp.11-13,pp.15-17):

> *""secure by design" shifts the focus of security analysis on architectural designs instead of implementations and outcomes. When deployed to research, this approach aims to identify whether security problems are merely symptoms of an architectural problem."*
>
> — *Klooz 2021 p.12*

A related formulation from the "secure by design" authors is:

> *"secure by design shifts the main focus of software assurance from finding security bugs to identifying architectural flaws in the design."*
>
> — *Santos and Tarrit and Mirakhorli 2017*

My 2021 thesis elaboration develops and explains the use of "secure by design" as research approach in socio-technical research, including some illustrations:

> *"Security by design explicitly, inherently avoids single points of failure/tampering/deception to affect the whole system. At first glance, the separation of powers fulfills that role in non-technical social architectures.*
> *Security by design is strict, does not tolerate exceptions, and considers all potential conditions of the given institution and architectures (complementary institutions, input, output): it considers all intended and predictable states/conditions institutions and architectures can adopt. Concerning unintended/distorting behavior/input/output, compensatory behavior of other institutions has to be in place: compensating institutions for explicitly avoiding predicted unintended/distorting behavior, and an architecture that implicitly avoids unforeseen behaviors to affect the whole system. Unavoidable vulnerabilities/behaviors must be identified so that the system is not used for something where it does not provide security.*
> *Especially the implicit automatism challenges the separation of powers, where each separation needs to be activated by intention without automatic activation when in a relevant state: the affected party must know about it to activate a balancing power. Therefore, if information is available, the separation is secure by design. Thus, the separation is secure by design in cases when the affected party knows about the issue.*
> *However, it is not secure by design if the balancing power cannot evidently solve the issue, e.g. due to lack of evidence. Thus, secure by design is also limited by the information available to the balancing power and therefore, limited to issues that are sufficiently documented with verifiable information. Nevertheless, this already includes many critical real-world issues, where the separation is secure by design. However, a separation of powers does not consider IT and socio-technical aspects, which are today related to each power and each affiliated entity."*
>
> *"A distinct case, which was already considered in several governments, is the goal of achieving security by implementing government IT in a distributed blockchain. However, if one administrator of one power administrates the chain, the password of this one administrator can manipulate the whole chain, bypassing any separation at this level."*
>
> *"The concept further complicates when multiple layers run on top of each other. Blockchain applications can be secure by design: they run on top of an operating system (OS), exposed to their behavior. However, one system cannot distort the blockchain. In Bitcoin, it would require more than 50% of all systems' computing capacity to distort the blockchain's state. The European Union runs on top of its sovereign states' systems, within the international society. However, each member state can distort the state of the system by vetoing decisions."*
>
> *— Klooz 2021 pp.15-16*

So when "following" the problem solving, secure by design will be used to identify single points of failure and points of potential unintended input/output in the socio-technical architecture, which could be intentionally exploited or cause accidental issues. An obvious example would be a point where an individual entity or institution could facilitate that a security flaw gets not processed and solved at all without others being able to intervene or to become aware, which

excludes an appropriate response to such an outcome. Unintended input can further foster intentional exploitation of such a single point of failure, and unintended input can apply not only to technical but also to social institutions and people (unexpected input by itself can cause irrational behavior and distraction of people).

2.2 Analysis & stages

In this project, I will predetermine in advance different "problems" (with different CIA security types; both time & non-time critical problems) that had to be processed and solved by the Linux system (socio-technical Linux system, not only the kernel or an operating system).

While following the flow of the problem processing, I will determine and map the problem solving with the social and technical institutions that are passed and that facilitate/foster the problem solving. Also, I will connect the institutions to their respective formal frameworks (=formal actors) that provide and impact these institutions (e.g., Linux kernel organization, Linux Foundation, Red Hat, ...). These frameworks form one limitation of the research because they are on themselves not analyzed against their own environment, which would introduce further interrelations.

After the major analysis, I will determine the competition and collaboration around these institutions to capture the dynamics of problem solving and to capture the factual "conflict actors". For example, at one place, a debate around a problem might formally involve representatives from the formal actors “Linux Foundation”, “Red Hat” and “Arch Linux”. However, in fact, the “conflict actors” in the competition of the problem might compete around a performance-focused and a security-focused solution, whereas representatives of all three formal actors appear in both “conflict actors”: employees of all three are found in the performance-focused actor, and employees of all three are found in the security-focused actor. So competition takes place, but not between the formal actors but between "conflict actors" that are determined by the immediate conflict itself.

Further, even if a formal actor has a role by formal statutes, this does not prove that this actor fulfills that role in practice (so, the formal actor can be absent in the factual “conflict actor”), and vice versa. Additionally, this stage will keep evaluating if the analyzed socio-technical architecture fulfills the secure by design criteria throughout problem processing or if there are potential blind spots.

An institution-based approach is more resilient to biases, and it facilitates to clearly identify the factual “conflict actors”: institutions facilitate competition and collaboration at one place, bringing all (formal actors, individuals, machines) entities that have these institutions in common together, allowing them to solve the problem by also using all the other institutions in their respective repertoire. However, if there is one solution (for the current stage of problem solving) that all have already in common, the current stage contains collaboration only.

If there are different potential solutions, there is competition among some institutions that are not shared among all but that form potential solutions, so that "conflict actors" can be derived and distinguished by their equilibria of competing not-shared institutions: e.g., the tests in the Fedora build system might be shared institutions among all and it might be the place for all to come together to find solutions for problems that have been identified there, but one “conflict actor” might be interested in deploying additionally a restrictive technical institution (e.g., code) to solve the problem, whereas the other “conflict actor” might favor a social institution by making user space developers aware by warning messages and documentation in order to offer them flexibility.

A general rule for my project: institutions make up the agency of actors, but institutions have no agency without actors. Complementary, the natural selection of institutions is more indicative than the natural seleciton of entities (Klooz, 2021, p.31): if an institution is competitive, the entities that adopt the institution will prove competitive. Entities who reject the competitive institution will not prove competitive in the given competition, and thus disappear with their non-competitive institutions.

Therefore, because institutions make the agency of actors, it is less relevant for the occurring agency which actors are competitive but mostly which institutions these actors contain/deploy. When put in contrast to the "conflict actor" concept, the "conflict actor" is obviously derived from common and competing institutions, which makes the "conflict actor" concept already incorporating the focus on the natural selection of institutions. Yet, it has to be kept in mind that not only actors can replace their institutions (and thus, their agency) without changing their names, but also institutions can be re-interpreted over time, keeping their name but creating a different agency (e.g., "sovereignty" has been re-interpreted over time; ibid., p.5). In short, the institution is changed, but the name remains.

The existence and absence of institutions, having them in common or not, offers new ways of comprehensibly describing the dynamics of collaboration and competition, and their "conflict actors": this approach does not assume that a "conflict actor" is an "obvious constant" but allows the “conflict actor“ to change in the dynamics of competition and collaboration, determined by institutions. These institutional dynamics’ actors can be comprehensibly distinguished by their institutional equilibria. In this stage, the goal is only to capture the dynamics of competition and collaboration of and around identified institutions, and derive their "conflict actors" (which indirectly also documents the dynamics of conflict actor development). Deeper analyses of these two types of dynamics or analyses of the conflict actor development is not intended.

After these analyses, I will check if and where identified formal and "conflict actors" and/or institutions are state institutions and/or (fully or partly) state actors ("state" in terms of being a public entity belonging formally to a sovereign state; e.g., NIST belongs to the United States' department of commerce). This is only a simple determination and no sophisticated analysis. Nevertheless, it can illustrate if and where we can differentiate between the "International Society of Sovereign States" (originally-intended for social security provision) and the system around Linux (informally doing socio-technical security provision), and if and where these systems are overlapping and/or blurred.

I will use the theory of International Relations when sovereigns have to be distinguished for a given area/field: societies shift power to (and reduce their freedom against) their respective "sovereign states within the International Society". Sovereign states act in anarchy to provide order and security to their respectively subordinated society. However, if the Linux system is providing types of security where no one else does (=provide security in anarchy, or "develop the anarchy"), it could be argued that the Linux system achieved sovereignty in this definition. Yet, in anarchy, both systems can (and are are likely to) be overlapping, interrelated and interdependent, making both the "sovereignty argument" of the recent sentence, but also the existing literature’s presumptions of "sovereignty" and "sovereigns" imprecise and potentially misleading.

Finally, I will compare the publicly known attack vectors and vulnerabilities of the Solarwinds-Hack 2019-20 against the determined socio-technical Linux architecture: do we have indication that institutions are in place in the socio-technical Linux system that provide resilience against such attacks?

Yet, this part of the project will be only indicative and cannot prove anything because of limited (and potentially selective) data that has been published, limited reproducibility, and the comparison of two systems that are comparable only to a limited extent (and that are maybe even overlapping). However, on the other hand, the reason for the limited comparability can also be the reason for their different resilience: the system around the SolarWinds-Hack is constituted within the international society by contracts among formal actors and clearly subordinated to sovereign states (especially the US) including when it comes to investigations.

Contrary to that, the transnational socio-technical Linux system is constituted by common problem solving institutions around code and information flow: this system is constituted by the product (which is more than a deployable kernel), and not vice versa. Although the Linux system can contain and involve state institutions and state actors, these have not yet proven dominant or sovereign against it. However, persistent or temporary overlaps and blurs of systems (further fostered by their different type of constituting) are possible and likely, and can make arguments of “who is dominant against whom” imprecise, as already indicated above.

3. Going beyond the surface and determining appropriate "problems"

3.1. Beyond the surface of problem solving

When evaluating problems that are eligible for this project, it becomes obvious that the socio-technical Linux architecture comprises competition already at the level of entering the architecture: problems are not allocated to an entity to solve it. Instead, anyone can create solution candidates that are then put forward as *merge requests* into the kernel. Also, problems have not to be published by the Linux kernel community: anyone can propose solution candidates even to problems not yet considered.

From an institutional point of view, it can be said that once anyone starts to develop code according to the requirements of the kernel community with the goal to successfully add it to the kernel, this belongs already to the socio-technical Linux architecture. In this view, the success itself is only the outcome of the initial competition for code that takes place within the Linux system (a competition that could be highly relevant for Linux's capabilities). However, from the kernel-focused point of view, it is the successful *merge request* that makes something part of the architecture, which implies that at the time of the initial code development (which is before the *merge*), it is not known if it will become part of the kernel.

Yet, with regards to the institutional point of view, it could be already an important facilitator of the Linux architecture's resilience that there is competition that takes place against *merge requests* into the kernel (which are well complements for a developer's reputation and well to gather competitive experience and knowledge) based upon what is technically possible and what seems technically advantageous: competition does not go towards current problems that have been already identified in societies, but towards what possibilities of code could be exploited next: any problem solution for any problem anyone has identified can be tackled, even without explicit demand and even if the problem is not yet apparent or even the problem is not yet existent at the time of *merge* ("competition towards tomorrow").

On one hand, this "separated competition" before merges into the kernel creates the possibility to create imitation by proposing what could be deployed next and thus, put it into the global socio-technical competition so that it can prove competitive there in order to get deployed globally (societies adopt what code provides if it proves competitive): the demand of global socio-technical societies can come with the product being offered, not only vice versa (yet, societies provide incentives and can reject solutions).

On the other hand, it leads to two different but complementary competitions: the competition towards kernel merges already solves problems that have not yet been considered by (and that have not yet impacted) the ongoing global socio-technical competition in socio-technical societies and their organizations. At this point it is possible that this is linked to the success and increasing spread of Linux but also its resilience and its capabilities.

Therefore, a "constant" in the kernel merge competition is what value code can provide immediately next after the last *merge*, and not only tackling immediate demand of a socio-technical society (although the latter might be another "constant"). The kernel merge competition's outcome then enters the competition of the socio-technical societies, which potentially adopt (but are at the moment widely limited to) what the Linux kernel then provides: Linux is law.

A conspicuous difference between the kernel merge competition and the global socio-technical competition is the quality of demand: the demand of the Linux community is exploitable high-quality code that has to undergo a rigorous selection and is assessed globally by qualified and well-reputed developers and scientists: what enters the global socio-technical competition has already proven to pass their assessment successfully if the Linux system is the origin of what enters. However, the global socio-technical competition often makes (or even put pressure on) people and organizations to adopt technologies and means that are not well understood by many or even the majority of users.

The latter then formulate their demand in a way that facilitates and fosters low quality supply, often intuitively by their behavior and not explicitly: yet, what Linux puts into the following competition has already proven to be a high-quality product (at least, in its intended circumstances).

If there is a relationship between the two competitions and if the resilience of the socio-technical Linux architecture (and the factual impact of this relationship) is a realistic possibility so far. Yet, it is unlikely to be proven within this project. For sure is only that the competition towards *merge requests*, which have to pass qualified developers, fosters the efficient identification of what code can provide and of the provision of corresponding and appropriate supply (it is competition-based: unlike command economy-type allocation of both demand evaluation and supply provision).

3.2. Preparing the problem determination

To make the research comprehensible and to make it applicable to both definitions (the institutional point of view and the kernel-focused point of view, as elaborated in the previous *"Beyond the surface of problem solving"* section), the project will focus on problems that have successfully entered the kernel. Although being part of the socio-technical Linux architecture from an institutional point of view, unsuccessful *merge requests* are beyond the scope of this project.

As proposed, appropriate “problems” have to be identified to reveal as much of the related architecture as possible. The analysis shall “follow” and evaluate pre-determined “problems” ("follow" from the appearance or proposal, whatever occurs first, up to the solution) that are already at the beginning of “problem solving” considered security-relevant for CIA: time-critical and non-time-critical problems (e.g., the SELinux mandatory access control was at least from a security-perspective a non-time-critical demand, unlike a critical security vulnerability). Additionally, problems that are not linked to security once their “problem processing” begins have to be identified as well: indeed, especially if problems are not explicitly linked to security by their demand (which can relate to the demand of both elaborated competitions, including the demand for exploitable high-quality code in the kernel), these problems can be dangerous because competitors can be encouraged to neglect security considerations to keep or gain competitive advantage: Linux is competing against alternatives as well, but it seems to prove both sufficiently secure, and competitive with most market shares being persistently increasing.

So, the predetermined problems have in total to cover these conditions: CIA, time-critical & non-time-critical, security-relevant from the beginning and not security-relevant from the beginning. Generally, the presumptions of the problems (especially their classifications as, e.g., time critical or security relevant) form a limitation (or, bias) of the research. The limited number of problems that can be processed in one MSc project are another limitation.

To reduce the likelihood of mixing potentially-different socio-technical architectures of different Linux eras (which would be a blurring factor), I will limit the project to problems whose problem solutions have been *merged* in the kernel "5" era (5.0 - 5.19.17: March 2019 - October 2022) (kernel.org, 2019; 2022).

3.3. The "mainline" and "stable" branches of the Linux kernel, and understanding their "commits", "merges", "backports" and "branches"

To make the following sections and issues like *merge requests* comprehensible, it needs in advance an explanation of Linux’s technical *version control system* as far as necessary. The Linux kernel is developed and organized using the version control system *git*, which is itself a major institution of the socio-technical architecture of Linux, impacting the latter’s socio-technical agency. Indeed, *git* was originally developed for the development of the Linux kernel (originally developed by Linus Torvalds), and the development of *git* and Linux remains interrelated.

When joining the kernel community, developers have to *clone* the Linux kernel "repository", preferably from *kernel.org* (this is getting a local copy in which they can work). Sometimes *cloning* is also named with the more generic *forking*, although the term "fork" is used also in other ways. Changes/additions/removals (hereinafter all three are summarized as "changes") to their *forked/cloned* "repository" are *commits*: one *commit* can contain multiple changes in multiple files. *forked/cloned* repositories are hereinafter named local repositories (of the respective developers).

Once developers have consolidated coherent/cohesive changes in one *commit* that changes the *state* of their local repository, the outcome can be requested to be *merged* into the related superordinated repository (at the initial stage, *merges* are requested using commands/structures in "mailing list mails", which can be generated by *git* and which differs from GitLab *merge requests* or GitHub *pull requests*). However, *pull requets*/*merge requests* (both can be assumed equal in this report) which are comparable to GiHub/GitLab might be used at later stages. There are usually multiple instances (= multiple *merges*) until the last *merge* finally ends up in a repository that is used to build "production" kernels. From one repository to the next, it is likely that each time multiple *commits*/*merges* are consolidated towards the next stage up to a final repository that is used to build "production" kernels.

After a developer suggested a *commit* by forwarding it from his/her local repository to the mailing list requesting its *merge*, the next repository in which the new *commit* might be *merged* to can be from another developer (e.g., the responsible maintainer of the related *sub-system* the *commit* will belong to) or a related repository in *kernel.org* or in some cases somewhere else. Responsible developers, such as *sub-system* maintainers, might have their own local copy to work in and a repository in, e.g., *kernel.org*, which is their public repository. Any change in any repository will be highlighted by what is added and what is removed (everything with *commit messages*) when two repositories are synchronized with each other: including when the maintainers synchronize their local repositories with their *kernel.org* public repositories, but also when anyone clones the public repository for, e.g., development, or if the next stage's maintainer *pulls* new *commits* from lower *sub-systems* to consolidate their *commits* at the next stage. This illustrates the later stages' capability to also do *pulls* instead of waiting for *merge requests*.

At the initial stage, each *commit* is its own mail to be discussed in the mailing list. A *merge*/*pull* can contain one or more *commits*, and one or more other *merges*/*pulls*. A *commit* can contain multiple other *commits*.

A repository can have multiple *branches*. E.g., a *branch* for each version (helpful, e.g., if multiple versions are maintained and updated concurrently) and one or multiple development *branches*. *Branches* can be *merged* into each other for equalization, but individual *commits* can be also transferred among them. If *branches* are changed in ways in which they loose compatibility, they have to be *rebased* to each other. However, *branches* can become completely incompatible to each other, e.g., different *branches* for different versions that are maintained concurrently might be incompatible at all, being not "rebaseable". Of course any repository at any stage at any place (local and public) is itself a normal *git* repository and can thus technically get *commits* directly from entities with corresponding privileges. Developers should *pull* from time to time to get the changes that have been introduced to a repository/*branch* since in order to ensure that they do not develop against obsolete states of Linux. *Fetching* and *remote branches* shall not be considered in this project.

3.3.1. How Linux organizes its git repositories

In case of the Linux kernel, *commits* can be discussed in the mailing lists once put to discussion there. *git* supports putting *commits* to the mailing lists in the corresponding way and with the *commits* being signed using *OpenPGP* (this takes not place using email clients). The kernel repository has a *branch* for each kernel (which is, each version: e.g., 5.10). As long as a kernel remains maintained, each "update" of the respective kernel adds a third number: e.g., 5.10.2 is 5.10 after its second update. However, 5.10.2 is only a *tag* within the 5.10 *branch*. A *tag* only marks the specific *state* of the *branch* at a given time. All currently maintained kernels are so-called *stable* branches. E.g., 5.10 is a *stable branch*.

Concerning the development of the future (so, the next) kernel, there are time slots in which it is possible to conduct *merge request* that bring new features in the so-called *mainline* branch (Kernel Docs, 2022a), which is the *branch* in that the future/next Linux kernel is developed. Once a time slot is closed, this kernel gets no further features. However, the latter does not mean that this *branch* will not be changed until the next time slot: the accepted *merge requests* and the resulting new kernel have now to be evaluated and tested, and might get further *commits* to fix issues that appear during testing. Due to the complexity of such technologies, further adjustments and bug fixes are likely before this kernel can be released. Therefore, the next kernel passes usually several *release candidates*, such as 5.10-rc1 and 5.10-rc2 before the final 5.10 is accepted and released. *Release candidates* are *tags* in the *mainline* branch. Once released, the new kernel becomes a new *stable* branch, whereas the *state* of the time the new *stable* kernel is *branched* from the *mainline* branch is *tagged* within the *mainline* branch. Then, the development of the next kernel will be started in the *mainline* branch, whereas the new *stable* branch can be used to build "production" kernels and it will be updated as long as it is *maintained*. The *linux-next* branch is not relevant for this project.

There are several maintained *stable* branches: on one hand, the latest released kernel and in some cases its predecessor (in case of, e.g., major changes, more might be maintained because there is no rigid rule). On the other hand, the so-called *Long Term Supported* kernels. All of them receive updates against flaws and other issues as long as they are supported/maintained. So, all *stable* branches that remain maintained can receive *commits* if they are affected by a flaw or another issue. This implies, one *commit* might be added to multiple *branches*. *Long Term Supported* kernels usually do not receive new features but only bug fixes and other corrections that improve stability and security. However, because *Long Term Supported* kernels are usually maintained for years, there might be exceptions for features that, e.g., have proven reliable and that have evolved to common demand.

3.3.2. How the kernels evolve

If a bug (or another type of flaw) is discovered in any maintained kernel, it gets fixed by additional *commit(s)* which will take place as long as the related kernel (= its *branch*) remains maintained. If the *commit(s)* that caused the flaw are also contained in older *branches* as well, the *commit(s)* that fix the bug/flaw are *backported* to all affected *Long Term Supported* branches. The question if *backports* will be necessary has to be considered early in order to develop fixing *commit(s)* that are as easy to *backport* as possible. If a flaw is discovered only in one of the maintained *branches*, it will be fixed only there.

Each repository is a *tree* of *branch(es)*, whereas the *branch(es)* are *trees* with *commits*. *Trees* can be *merged* with each other if they are generally compatible or if they are at least *rebaseable* to each other. The *trees* keep any *state* of any *branch* of any time, and they remain able to show any *commit* of any time including its place(s) in its *tree(s)* and show the developments over time of and among *commits*. This makes it possible to, e.g., identify and review or even revert *commits* long after they have been added (as noted, *commit(s)* go along with *commit messages* that describe the content of the *commit*). The name of any *commit* is a hash value that is generated from its content and its preceding *commit* (its "parent"): at this point it becomes obvious that the *trees* are *hash trees*, which form chains of hashes. It might be additionally noted that there cannot be a *commit* that does not belong to at least one *branch*.

3.3.3. Linux kernels in the context of the project, and probabilistic or blurring factors

At this point it might be noted that at the time of writing, there are still *Long Term Supported* kernels from the 5 era. But as indicated before, these kernels only receive fixes for bugs and security flaws from which they are affected. This project will only analyze "problems" that have been already solved. The *trees* will be only considered up to the date of the latest *state* of the latest non-*Long Term Support* kernel of the 5 era, which is the date of the 5.19.17 release (which marked the *end of life* of 5.19). Later developments of the *Long Term Support* kernels of the 5 era (5.4.x, 5.10.x, 5.15.x) after that date will be ignored as well. It cannot be explicitly proven that this is not a limitation, but the likelihood of this impacting the results of the project is unlikely. Additionally, it might be noted that it can be probabilistic if a *commit* contains new features or only bug/flaw fixes.

Furthermore, there are vendors like Red Hat that keep maintaining some *stable* branches for longer terms themselves, even if these *stable* branches are no longer maintained by the Linux kernel community and even if these *stable* branches have never been designated *Long Term Support* by the Linux kernel community (e.g., Red Hat's 5.14.0-162): these kernels will be not considered in this project.

Additionally, it might be noted that the terms used in the kernel's problem solving are often focused on the actual task/outcome and not on *git* operations. This links to the term *patch*: any change to a kernel is a *patch*. E.g., if a kernel has a bug, a *patch* is developed to fix the bug. This *patch* is developed and introduced to a *branch* with a *commit*, and the *patch* can be "transported" throughout multiple *branches* through multiple, potentially different, *commits*.

git is an omnipresent institution of and around Linux: each line of code of the Linux kernel is managed through *git* and its documented and long-established means: all developers need to use *git* and adapt to Linux's *git*-based repository organization. Indeed, *git* was developed originally for the Linux kernel and later spread to other projects. It is a fact that the Linux kernel source is a *git*-repository, with all of *git's* capabilities and limitations. Therefore, that *git* is a major institution of the socio-technical Linux architecture is not questioned in this project and presumed in the later analyses.

3.4. The problems that are to be "followed"

When it comes to security-relevant and time-critical problems, I will analyze two recent fixes of the kernel 5 era that solved CVE-numbered flaws. Yet, it might be noted that the classification for security-relevance in kernel issues does not imply CVEs, and vice versa: the Linux kernel security team (which might be analyzed itself in later stages) decides themselves if something is security-critical or not (kernel.org, 2020b). Even if something is classified as highly security-critical by the Linux kernel security team, this does not automatically lead to a CVE request. However, if the reporter of an issue has requested and received a CVE number before the related *commit* has been done, the CVE number shall be added to the *commit message* to have the relationship comprehensibly documented. In order to distinctly and comprehensibly fulfill as many "security-relevance" definitions as possible, time-critical and security-related problems have to be issues with CVE-relation.

The two most recent CVE-related fixes at the time of this determination, which affect 5-era kernels, are *CVE-2022-40768* and *CVE-2022-3435*. Additionally, there are several interrelated WiFi-CVEs with interrelated *commits* and discussions. To keep the results comprehensible, distinct and to avoid predictable blurring factors, the WiFi-CVEs shall be excluded due to their obvious blurred interrelations and overlaps with each other and in their *commits*. *CVE-2022-40768* and *CVE-2022-3435* are independent and distinct issues without other CVE interrelations.

When it comes to security-relevant but non-time-critical problems, I will analyze *wireguard* (kernel 5.6) and *adiantum* (kernel 5.0): both are already widely deployed, have proven competitive in one or more use cases, and both are features that have an impact on all security types: CIA. Also, both are explicit security features and used not only for internal use within the kernel, but security features that are to be used by users or external tools to provide security that goes beyond the respective kernel space: these two shape agency beyond the internals of the kernel, and both are not bound to the kernel (e.g., there are already many user space implementations also for other operating systems especially for *wireguard*). In the 5 era, these two are the earliest two which fulfill all criteria. That they are the earliest ensures the longest possible period in the 5 era in which subsequent *commits* can (or have to) change or fix issues of these two technologies, in order to evaluate how these technologies develop over time (e.g., do they cause themselves CVE issues that need time-critical fixes).

Adiantum is a disk encryption mode that is already widely deployed on Android but generally available on the Linux kernel: besides its improved security features in comparison to the dominant AES-XTS (e.g., true wide-block encryption; Biggers, 2018), it enables disk encryption where it was practically not competitive before: on platforms that have no AES hardware instructions (which is a hardware implementation of AES), AES-XTS and AES in general are slow and their corresponding performance and power consumption can pose themselves a threat to the security of availability,

especially on mobile devices. Until the introduction of *adiantum*, disk encryption was not competitive (and not obligated by Android in most cases) and thus rare on devices that depend on long battery life but that have no AES hardware instructions. Therefore, disk encryption (and every service that depends on it) was not available for many use cases: many mobile phones/smartphones, tablets and single-board computers (such as all Raspberry Pi models to this day) do not have AES hardware instructions.

Wireguard provides secure communication and can be embedded in and used for many different types of solutions in both kernel and user spaces. It can transport both TCP and UDP traffic. Just like *adiantum*, its deployment is also realistic on devices that depend on long battery life and/or that have no cryptographic hardware instructions. More generally, it is designed to be cost-efficient, use minimal resources on all hardware, and to have a minimal overhead (unrealistic to achieve with TLS-based solutions): in short, ensure that achieving CIA has as less negative impacts as possible. It is intended to be easy to deploy and maintain in operations without the need for alignment of affected environments or services (unrealistic to achieve with IPSec due to its complexity and its restricting protocols). Additionally, it natively contains and enables the use of two different keys, each securing all data on itself, with one being able to remain resilient also against quantum attacks: a symmetric pre-shared key and an asymmetric Curve25519-derived private key. The use of the pre-shared key is not obligated.

Non-security-relevant problems are probabilistic: everything is somehow linked to a type of security. In this project, "non-security-relevant" is about issues/problems that are not considered and not treated as "security" problems in their processing: this might offer a complementary different perspective on the Linux architecture, and potentially revealing more institutions that themselves shed a different light on the role of "security-focused institutions" (and of course, what institutions might be "security-focused" and what not). This might indicate how far the treatment of security-relevant problems differs from non-security problems, or how far "security" is integrated in terms of being no "dedicated topic" with security and non-security problems being treated equal.

To focus on problems whose potential impact on types of security is comprehensible, I determined *Freesync* (non-time-critical; kernel 5.0), which is not immediately security-relevant but a highly complex technology that, being part of a graphics driver in the kernel, has access to a lot of kernel resources, which cannot be limited if it shall be able to fulfill its tasks. On one hand, at the worst case, flaws in this complex technology could be exploited to gather control of the kernel. On the other hand, if a graphics driver breaks in some circumstances, it can cause risks to the security of availability.

Also, *Freesync* was added immediately at kernel 5.0, which means that the analysis can track developments throughout the determined era: so, to track the development and impacts of *Freesync's* initial *commit(s)* for several years to evaluate if these *commits* cause problems, which thus make further *commits* necessary to resolve these problems (the same analysis as intended for *adiantum* and *wireguard*).

Because the CVE-problems are themselves only fixes of existing technologies whereas their *commits* are limited to the immediate fix, tracing their subsequent development is considered not relevant for this project.

Due to "following" already bigger problems (*wireguard*, *adiantum*, *Freesync*) throughout the 5 era which are likely to already identify further *commits* for analysis in their aftermath, another dedicated analysis of a non-security-relevant but time-critical problem can be seen as useless. However, for the unlikely possibility that no non-security-relevant but time-critical problem will be covered through following *wireguard*, *adiantum* and *Freesync* throughout the 5 era, a problem has to be determined in advance to the analysis in order to ensure that the selection is not biased by the then-preceding research. Therefore, a non-security-relevant but time-critical problem has to be selected at this point, but its analysis will be only implemented if the development of *wireguard*, *adiantum* and *Freesync* over the 5 era does not cover non-security-relevant in combination with time-critical.

Because the non-security-relevant but time-critical problem is primarily intended to be compared against the two CVE-related problems, it makes sense to select it from a kernel that is timely-close to above CVE problems. Also, the time-critical non-security problem should be from comparable realms of the kernel like the CVE problems (theoretically, at the best: differ only by the security / non-security feature), in order to more clearly capture the differences of security and non-security. Therefore, the selection of *"mmc: core: Terminate infinite loop in SD-UHS voltage switch"* from 5.19.15 makes sense: this bug fix and the CVE-2022-40768 above both are related to memory.

The bug fix relates to the handling of "SD" memory cards, the CVE-2022-40768 on the other hand relates to the Small Computer Systems Interface protocol, which is used to connect and transport data among storage devices. It has to be clear that at this low level, these two issues are far away from being equal, and can prove to be not even part of the same "sub-system" of the Linux kernel. However, their impacts for CIA in their respective use cases in "production" can be comparable. However, as elaborated, the *"mmc: core: Terminate infinite loop in SD-UHS voltage switch"* problem analysis will be not implemented if any of the *wireguard*, *adiantum* or *Freesync* analyses already cover its area.

Generally, the "problem" analyses will also consider the following stages after *commits* have been added to the related Linux kernels: the testing stages have just begun at this point, and the massive distributed testing around the Linux community and related communities (especially testing on deployable operating systems) is to be conducted after the *commit* has been added: this includes the feedback loops with relevant information that flows back to the kernel development.

4. Time-critical and security-relevant: the kernel to tackle CVEs

4.1. CVE-2022-40768

4.1.1. Starting the problem evaluation

The abstract of CVE-2022-40768 is: *"drivers/scsi/stex.c in the Linux kernel through 5.19.9 allows local users to obtain sensitive information from kernel memory because stex_queuecommand_lck lacks a memset for the PASSTHRU_CMD case. "* (Mitre, 2022a).

Although the problem was identified and determined for this project through the ChangeLog of the 5.19.16 kernel (kernel.org, 2022a) and from the kernel mailing list archive *lore.kernel.org* (kernel.org, 2022b), it makes sense to start the actual research of this "problem" from the CVE page to get an impression from the CVE perspective.

The CVE page (Mitre, 2022a) already indicates that multiple entities are involved at managing the CVE in the kernel, and that tackling the tasks around this CVE is distributed: at first glance of the CVE page, the involved entities are so far:

- The *Fedora Project*: a Red Hat Inc.-sponsored community that, among others, develops and maintains *Fedora Linux* (Fedora, 2022), with Red Hat being itself the company that develops and maintains *Red Hat Enterprise Linux* (which is derived from Fedora Linux) and other software (Red Hat, 2022),
- The *Debian Project*: an independent community that, among others, develops and maintains *Debian Linux* (Debian, 2022a; 2022b), being associated with the *Software in the Public Interest* non-profit corporation that has risen from the Debian community to support, among others, Debian's goals (Debian, 2022a; SPI, 2022),
- *kernel.org*: the major domain/page of the *Linux kernel community* to develop, maintain, discuss and report issues around the Linux kernel, being provided by the *Linux Kernel Organization, Inc.*, a non-profit corporation that is intended for this purpose (kernel.org, 2022c), which includes the related services: among others, the major *git*

instance of the Linux kernel (git.kernel.org), the kernel.org mailing lists (multiple sub-domains; kernel.org contains not all mailing lists for the Linux kernel) and the kernel.org mailing list archives (lore.kernel.org) (although the mailing lists are archived under multiple domains on the Internet),

- The *Openwall Project*: a security-focused community centered around and led by the Russian security specialist Alexander Peslyak that, among others, develops and maintains *Openwall GNU/*/Linux* (Openwall, 2022a) and that also organizes paid support services for its free and open source software (Openwall, 2022b).

4.1.2. Determining the omnipresent institutions

With all the CVE-related links and references on the Mitre (2022a) page, it can be already said with regards to all these domains: Linux is not just the low level foundation of most servers on the Internet and thus of many *SSL/TLS*-based services (Secure Sockets Layer / Transport Layer Security), but also depends partly itself on these services: much documentation and archiving is provided using *HTTPS* (Hypertext Transfer Protocol Secure).

Much discussions take place on *mailing lists* (email), whose discussions are then published using *HTTPS*. The mailing lists contained on the pages that are known so far are all open so that everyone can contribute (= send mails: e.g., Webmail on HTTP(S), Simple Mail Transfer Protocol with or without SSL/TLS/StartTLS). Once signed up for a mailing list, people get the discussion contributions through mail (= receive mails: e.g., Webmail on HTTPS, Post Office Protocol v3 & Internet Message Access Protocol with or without SSL/TLS/StartTLS). Therefore, in case of the mailing lists, *HTTPS* is only for archiving in multiple archives on the Internet (an archive can be created simply by subscribing to the related mailing list to then consolidate and publish what is subsequently received). However, archives can remain a major source for documentation and information provision (e.g., search queries through search engines), but also ensure *distribution* of independent archives with, e.g., many archives (and many users) being able to identify manipulations in individual instances.

The *HTTPS* services that have appeared so far are secure and fulfill all requirements set by current browsers. So far, all pages do implement either the revised *TLS 1.2* (not the original *TLS 1.2* which has an increased priority for *SHA1* when negotiating algorithms: this is already rejected by some browsers) or *TLS 1.3*. All have valid x.509v3 certificates. I assume at this point that this is representative for the *HTTPS* usage around Linux's problem solving. But I will keep checking pages that are related to the analyzed problems. If any related page will not fulfill security requirements around *HTTPS*, it will be noted explicitly at the very place.

Yet, mailing lists and emails in general are on themselves vulnerable technologies. However, confidentiality is not a problem since all problem solving discussions so far are public anyway. Nevertheless, ensuring integrity from the sender up to the mailing list cannot be guaranteed by email protocols on themselves.

Still, at this point it might be referred back that everyone can contribute to these mailing lists. Therefore, nothing that is suggested on the mailing lists is "automatically added" to the kernel but critically assessed and discussed once it ended up at the mailing list. Therefore, up to the point where code for the kernel is posted to the mailing list, mails that have been manipulated could only be used to undermine the author (although in an investigation, it is likely that protocols among the different mail servers could at least indicate the manipulation: user → mail transfer agent (MTA) → mail delivery agent (MDA) → mailing list).

Further, it has to be said that intentional manipulations like that are not easy since the attacker has to be in between two entities at the very time of the data transfer. Additionally, even if email protocols can on themselves not ensure end to end encryption between origin and the final target, authenticated encryption between the respective entities can be indeed ensured as long as there is no restriction on cryptographic use (e.g., SSL/TLS with SMTP). However, without further means, this introduces automatically that the author and the mailing list have to trust the known entities in between.

In the end, only the MTA and MDA could do such an attack reliably. But it has to be kept in mind that the return/reward of such an attack is unlikely to be relevant or to have a noteworthy impact.

Additionally, all mails with code in the kernel mailing list threads (kernel.org, 2022d) have been signed-off using *OpenPGP*, which is a widespread and well understood technology considered secure (just like SSL/TLS and HTTPS) that can add end to end integrity from the sender up to all mailing list recipients. Further, for mails that contain code, signing-off is mandatory to ensure integrity, which is elaborated in the related Kernel Documentation pages (Kernel Docs, 2022b): OpenPGP signatures are enforced. So far, the formal Kernel Documentation is not just theoretical but it is implemented practically on the kernel mailing lists that have been analyzed.

Even the cryptography of *OpenPGP* is explicitly considered: for signatures, existing *RSA* keys with 2048 bit are accepted as of kernel 5 era, but new *RSA* keys shall be more than 2048 bit, but generally Elliptic Curve Cryptography with *ed25519* or *nistp256* is preferred (ibid). For the very purpose, future attacks on *RSA* 2048 bit (or, 2048+) are not relevant because at any time it can be stopped to accept new *commits* with related signatures. A dedicated analysis about which cryptography is practically deployed by developers in their *OpenPGP* signatures (e.g., if *RSA* 2048 is still used by them) will not be conducted. But the analyses around the verification of signatures before new *commits* end up in the kernel's *git* might offer indication in later stages of the research.

However, *OpenPGP* is used to sign *SHA1* hashes and the incompatibility between *SHA1* and *SHA256* makes it at the moment impossible to easily migrate the kernel trees. Therefore, the security depends in this case on both *OpenPGP* and *SHA1*, whereas breaking one also makes the other useless to the same extent. Yet, it has to be clear that despite the fact that *SHA1* is no longer considered a best practice and that it is suggested to be replaced for security use cases, it is not "broken for all use cases", or even "easy to be exploited".

In this use case, if a valid and trusted - e.g., *ed25519* - key is used to sign a *SHA1* hash of the content of a mail/message, an attacker can only exploit this by creating a malicious *message/code* that has the same *SHA1* hash value as the original mail/message that was signed by the valid key: finding message "b" that hashes to the same value "v" as message "a" is a second pre-image attack, and as of today, such an attack is not practically realistic against *SHA1*: especially because the content of the malicious message "b" cannot be fully arbitrary but has to contain content that makes sense, may it be a message that gives incentives to the mailing list, or code that puts malicious behavior into the kernel. This remains unrealistic against *SHA1*.

However, advancements in breaking *SHA1* are more realistic than for "best practice" algorithms like *SHA256*, and advancements might develop towards a more realistic attack over time. A debate about this is already ongoing and *git* was already prepared towards supporting *SHA256*. However, the migration has not yet begun, and a plan how to do the migration (given the incompatibility between *SHA1* and *SHA256*) does also not yet exist (Corbet, 2022). In the end, the risk at this point rises from advancements in breaking *SHA1*, and the absence of plans about how to quickly respond and replace *SHA1* in the case of new attacks.

Yet, given the analysis so far, it cannot be excluded if another *SHA1* issue can be exploited in the foreseeable future to achieve a type of *denial of service* in some way by forwarding or flooding with false messages/code through exploiting collisions in some way: this might become more realistic in the foreseeable future because some collisions in *SHA1* are already known (*SHA1* collision exploitation yet cannot be compared to the advanced collision attacks against, e.g., *MD5*). Yet, even if there are advancements in identifying *SHA1* collisions, this does itself not pose a risk to the "production" Linux kernels that are output by the kernel community.

Even critical security updates are unlikely to be hindered because of the massive *redundancy*, *distributed* responsibilities and *distributed* processes at *SHA1*-related processes: even to cause confusion in realms that are used to the fact that anyone can contribute and that are correspondingly capable to balance such issues to some level, it would be necessary to have this attack done by someone who is known and who has some responsibility (e.g., a maintainer of a *sub-system* of the Linux kernel) to make the attack start or proceed in later stages. Besides the lack of advantages and

the limited and compensable impact, this could be soon linked to the very person. Because of the obvious possibilities to compensate and the not yet existing possibilities for related *SHA1* attacks, such *DoA* attacks are out of scope for this project.

Nevertheless, the *SHA1* related discussion on *lwn* (Corbet, 2022) might indicate another informal way of information exchange and discussions around the Linux competition: *lwn* is itself linked to the Linux mailing lists and used by maintainers (e.g., the maintainer of current *stable* kernels) to spread information that might be *distributed* and discussed by adding *lwn@lwn.net* to mails of the mailing lists, so that the related mail ends up as article at *lwn*. It might be noted that according to the website of the Linux Foundation, the founder and editor of *lwn* is member of the "Linux Technical Advisory Board" of the Linux Foundation, implying another interrelation. Additionally, its articles (and other such pages' articles) are mentioned as information source for contributors within the Kernel Docs (2022b; 2022d). This concept and *lwn* itself can be argued to be themselves institutions. Many other pages on the Internet archive information passively or get information actively by kernel developers for publishing and discussing, whereas all contribute to the *redundancy*.

4.1.3. The chain of trust and sub-systems in the context

According to the Kernel Docs, signing shall be used to provide trusted communication among developers (ibid), especially for the so-called *sub-system* maintainers. *Sub-systems* have to be set in context and analyzed on themselves since a *sub-system* is already introduced in the title of the CVE: it is easy to identify that *scsi* is a *sub-system* of the kernel. The signing with *OpenPGP* forms itself an institution that is indicated in the mailing lists (and code that is transported through the mailings). This is confirmed in the Kernel Documentation, and in conjunction with *git* and the *git*-based *sub-system* structure, *OpenPGP* enables another institution: the *chain of trust* relates to a point made in the related Kernel Docs (ibid): *"Trusting the developers, not infrastructure"* (ibid). This is complemented by a sentence that complies to the "secure by design" approach: *"the Kernel Archives project has been to assume that any part of the infrastructure can be compromised at any time"* (ibid). The Kernel Archives relate to the infrastructure of **.kernel.org*. However, so far other related domains that are used for mailings and related services follow the same institutions.

Additionally, the Kernel Docs also document an important information that is illustrated and implemented throughout the mailings: signing with OpenPGP and the *sign-off* procedure in the *sign-off* area of mails are different things (ibid; and the related kernel mailing list topics). *Sign-off* is just an area within the mail that contains meta-data of who did what and who was involved where. The area is then part of the content that is signed using *OpenPGP*. Over time, this area can accumulate several people that are then mentioned for being involved in, e.g., development or review. Everyone who receives mails is capable to verify the OpenPGP signatures with the contained messages including the *sign-off* area. This also implies that everyone can verify if any mail that has been signed by *OpenPGP* contains a manipulated *sign-off* area (such as the current signer removed or added someone else) because all past mailings with their respective signatures are available to be compared against (and everyone can implement to test this automatically). In this project, adding someone or oneself to the *sign-off* area is named "to sign-off", unlike "signing" for digital signatures.

Besides the more technical aspects of the previous paragraph, the role of the *sign-off* area from a wider socio-technical point of view and its interrelations in the Linux architecture will be analyzed later in a dedicated manner: on one hand, this is to keep focused on the framework formed by the *chain of trust* and *sub-systems* for now. On the other hand, it can be anticiapted that in later stages much more developments and dynamics of mailing list mailings will be available to more precisely determine the *sign-off* role in the Linux architecture.

It makes sense to have a look on the *sub-systems* that relate developers and maintainers to *git*: *git* makes it possible to asynchronously *merge* different developments into each other until everything comes together at the previously introduced *mainline* branch or one of the *stable* branches. A *commit* can pass both >=1 discussions and also >=1 *git* branches of different developers/instances (which implies many repository clones/forks that *pull* the *commit*) until it ends up in a released kernel: this is an asynchronous and heterogeneous tree structure. Besides the major maintainers

who finally release the *mainline* (Linus Torvalds) and *stable* (Greg Kroah-Hartman in most *stable* branches) kernels, there are *sub-systems* that have themselves their own dedicated maintainers who request to *merge* new *commits* from their *sub-systems* to the next stage (and who proceed requests from earlier stages) once they assume them to be ready in order to finally get *merged* into *mainline* and/or >=1 *stable* branchces. Also, the organization of and among *sub-systems* is not equal but heterogeneous. E.g., some *sub-systems* may have a common aggregation tree for both direct development and several *sub-systems* (Kernel Docs, 2022c) before their developments are *pulled* by the next stage, which introduces another stage for them: one example is the *tip tree*.

An example for processing a *commit* that is accepted and passed throughout all stages would be: the *patch* developer's clone/fork (in their local repository) → *sub-system* public repository → *sub-system* maintainer local clone/fork to test and verify → *tip tree* public → *tip tree* maintainer locally → *mainline* & affected *stable* branches → *mainline* and *stable* maintainer locally. However, *git* can be used in multiple ways and this can differ among developers, maintainers and *sub-systems*. Also, once accepted by the respective maintainer, the next stage is likely to *pull* from the previous public repository, not from the previous stages' maintainer's local clone/fork. However, technically all is possible: if people have the responsibility for the respective stages (= maintainer), they are trusted to properly to use *git* in a reasonable way, instead of trusting a standardized infrastructure or something comparable. Obviously, in practice, the discussions and the stages are likely to contain many more people (with their clones/forks) to test, review, discuss, comment before a stage is passed.

However, although the preceding is relevant for understanding the flexible structures and institutions, the *scsi sub-system* to which the very problem belongs to, does not *merge* into an aggregation tree.

With a focus on the locally stored *git* repository forks/clones: these are stored locally in order to work on and in them while each single change that is *pushed* to or *pulled* from other *git* instances will be shown and documented at each instance (including other instances *pulling* the changes later from any of the existing instances): e.g., everyone can see when *pulling* who added which *commit* with which purpose and with which change.

Additionally, it is up to the developer on how many *git* instances code is stored: Linus Torvalds seems to also mirror his *git* repository (which contains the *mainline* branch) additionally on GitHub (GitHub, 2022). Primarily, the major *git* instance remains https://git.kernel.org/ that serves the *major* developers and where code from them can be *pulled* and where they *push* code: the git.kernel.org-*git*-repositories are those used to derive releases. However, obviously, a lot of people and organizations have clones/forks of Linus Torvalds' repository and will receive each change with the corresponding information when *pulling* changes, and obviously, all have to tailor to the changes when preparing themselves changes. It is the same for the *stable* branches, which are in a separated but compatible repository on https://git.kernel.org/:

Each *stable* branch in its repository was at the time of its *branching* equal to a *tagged* state of the *master* branch of Linus Torvalds' repository. Therefore, the mentioned *master* branch is currently the *mainline* branch. *master* is the default branch of a *git* repository, and because *mainline* only needs *tagging* for organization, additional *branches* are not necessary there. Once the very state of *mainline* is *branched* as a new *stable* branch into the separated repository (which contains all *stable* branches), it starts to develop independently: it might receive some future *commits* and some other *commits* not, and some *commits* might be *backported* to it, which implies that the latter are no longer equal to the original *commit*. Yet, this structure is a Linux-specific institution because *git* offers multiple ways of implementing and structuring such projects. However, although at first glance, it might l like all *stable* branches have the same maintainer, a deeper look on the *stable* maintainers over time makes this more a coincidence that often, but not always, applies.

If a new *stable* branch is *branched*, the "current" *stable* maintainer takes the responsibility. However, when the *stable* maintainer changes, the underlying institution becomes more evident: the new *stable* maintainer takes the responsibility for new *stable* branches that are *branched* after the new maintainer took over. But older *stable* branches might remain with the previous *stable* maintainer, whose responsibility will decrease over time until the last of the old

maintainer's branches passes *end of life*. However, this is something between the new and the previous *stable* maintainer, but it makes clear that it is assumed that once someone picks up responsibility for something, this person shall not just leave but find someone else to do take over or at least offer a sufficient time frame before leaving so that the community can adjust.

However, how to find replacement and how to transfer the responsibility remains highly informal. These informal institutions ("stand by your responsibility"; "how to transfer responsibility") are widely stated on the Internet, and seems so far to be practically implemented on a widespread manner: this also refers to code at all (if someone adds, e.g., a new technology to the kernel, even if it becomes not a dedicated *sub-system* status). When reviewing the *git*-related organization, it gets obvious that *git* repositories are organized around their developers and not around *sub-systems* or other topics (related to *"Trusting the developers, not infrastructure"*). Even in the *stable* branches that are not named after their respective maintainer, the respective public *branch* is finally based on the respective maintainer's fork/clone (unlike *mainline*, whose repository is named after its maintainer "torvalds", which itself contains the *master* branch that is *mainline*). However, in terms of *sub-systems*, there can be multiple maintainers for one *sub-system*, who have to internally coordinate themselves informally. The person-centric focus loosely links to *agile development* methodologies.

However, in conjunction with the ChangeLogs I evaluated during the "problem determination" phase, the recent analysis gives further evidence for a major institution that already appeared before: *redundancy*. Indeed, this is an institution I already identified and elaborated during my first thesis (Klooz, 2021, p.41). Much *redundancy* avoids single points of failure, which can also balance unintended input/output at one place. Even when it looks that the kernel that fixed this CVE (5.19.16) is released at one place (indicating a single point of failure), the ChangeLog documents that several people from different companies and different communities tested this kernel (kernel.org, 2022a), and through the public and signed code, each entity that deploys this kernel can verify with secure cryptographic means that what it gets is equal to what all these entities have tested. In this specific case, the kernel has been tested and signed-off by 11 people (ibid): their related messages were signed (*OpenPGP*) by them, which developed to their names being mentioned in the *sign-off* area of the final ChangeLog.

Additionally, the mailing lists of the CVE page indicate that at least three further entities have tested the kernel the way it was released and compiled it based upon their preferences and tested it independently from each other through their own pre-determined, and at least in these three cases, public/transparent testing institutions: Fedora, Debian, OpenWall (Mitre, 2022a).

Indeed, different preferences of different Linux operating systems may reveal different behaviors and thus, make it much more likely to identify unintended behavior (bugs, security issues). It can be assumed that further Linux communities and companies have tested the kernel, but further elaboration of this is out of scope. Yet, it is evident that it is practically not possible to achieve control over all entities that do testing and search for issues, and it is impossible to foresee which tests and preferences are used/deployed by them: this poses serious challenges for attacks.

The kernel community and the communities of the operating systems are overlapping: at least one developer of the ChangeLog is also member of one of the communities (Fedora) based upon the email address that signed-off the kernel (kernel.org, 2022a).

However, first the Linux kernel community might be analyzed, and the operating system-related communities (Debian, Fedora, OpenWall) later. First, I have to bring all discussions and points that have been made on *.kernel.org together, sort them by date and relate them to each other. The kernel.org link on the CVE page is the first, with a first information and an initial *patch* proposal by the maintainer of most of the current *stable* branches: Greg Kroah-Hartman (kernel.org, 2022d). Since there is no previous public documentation available that documents the report from the reporter "hdthky" (ibid), it can be only assumed that the reporter informed the kernel community through a

confidential email to the *kernel security team* (at the best, encrypted, as requested by the *kernel security team* in such cases) (kernel.org, 2020b) so that the problem can be evaluated for immediate risks and temporary mitigation possibilities for the Linux operating systems before being published (not each problem can be fixed in a day).

Alternatively, many companies and communities that develop and maintain a Linux operating system (or one of Linux's *sub-systems*) offer themselves comparable possibilities to inform about security issues. Through the overlaps and interactions of the entities, this would likely have a comparable outcome than a report to the kernel security team. At this point, it is not possible which of the possible ways was used by the reporter to forward the problem to the kernel community. We just know the issue was known to the kernel community before a CVE was assigned: as noted in the Kernel Docs, CVEs might be tracked and noted in the related mailings if assigned before the discussion in the kernel community begins, but this CVE was assigned on 18th September 2022 (NIST, 2022a) while the discussion of the issue has already begun on 8th September 2022 (kernel.org, 2022d). This explains that the CVE is not tracked throughout the kernel discussions.

Obviously, before a *patched* kernel can be released (which is a kernel that contains a *patch* that fixes the problem), the issue has to be handled on the *sub-system* level in order to then pass the subsequent stages up to the released *patched* kernel. In the discussions that focused on solving the problem on the *sub-system* level, the mailings contained already in the first mail several people on CC that were involved immediately (kernel.org, 2022d): the CC contained kernel developers from different organizations (Linux Foundation, IBM, Oracle, SuSE) and the reporter ("hdthky"), including the maintainers of the *scsi sub-system* James Bottomley (IBM) and Martin K. Petersen (Oracle) but also Dan Carpenter (also Oracle), plus one mail from Lee Duncan (SuSE) who is one of the maintainer of *ISCSI* (this information is contained the MAINTAINERS file in the *git* trees). However, it might be noted that the *scsi sub-system* does not relate to the *scsi* interface but to the protocol. Since this is also interrelated with *ISCSI*, this might explain the overlap of developers from *scsi* and *ISCSI*.

After bringing all kernel.org-related pages for kernel 5.19 together that followed the initial kernel.org mail (2022d) (chronological references after 2022d: kernel.org, 2022j; 2022e; 2022f; 2022g; 2022h; 2022i; 2022n; 2022a), it is worth to note that the initial meta-data analysis identified that the maintainer of the *stable* branch Greg Kroah-Hartman was from the beginning involved, too (ibid): the first mail with information of the issue and an initial proposal for a kernel *patch* came from him. Additionally, in this case, the final *patch* (a revision of the initial *patch*) was contributed by the maintainer of the *mainline* branch Linus Torvalds. Discussion and contribution is not restricted. Complementary, the previously noted Docs pages contain pages to enter the discussions and how to contribute code.

With the assumption in mind (which cannot be proven explicitly in this project) that the reporter has reported through the intended channel over the *kernel security team*, another institution is indicated: "publication after *patch* draft if vulnerable and exploitable". Indeed, this ensures that any code which shall be added to the kernel undergoes the full, transparent and public processes. On the other hand, this ensures that at the moment of publication, a working *patch* already exists that could be applied directly by the Linux operating systems to previous kernel releases if the fix of the very problem is for a given Linux operating system more important than the finished processes towards official kernel integration: at least Fedora *backported* the last of the *patch* drafts (the one later added officially to the kernel 5.19.16) to the previous 5.19.15 kernel (they created a second 5.19.15-type kernel for their system, succeeding the original 5.19.15), not waiting for the official *patched* kernel, which was 5.19.16 (Fedora, 2022a).

It is not documented if there had been a discussion among related developers and how far they already assumed or knew this *patch* to be final. Yet, the tests on the Fedora operating systems have not been bypassed and the *patched* 5.19.15 had still to undergo the whole process of public automated and manual testing (Fedora, 2022n): determining the proper compromise seems to be shifted to the Linux operating systems that shall know their use cases and needs themselves. Additionally to the direct use of the initial *patch* draft as interim solution, the initial *patch* draft can be used to understand the problem and to maybe allow the Linux operating systems to find other ways to mitigate the vulnerability temporarily in their very "production" environments (e.g., disable vulnerable features of the kernel temporarily).

It might be noted that in this case, the *sub-system* maintainer was not actively involved in the discussion but only on CC throughout. The *commits* that introduced and transported the accepted *patch* (the revised "v2" from Linus Torvalds) from Torvalds' proposal up to the respectively final *commit* that ended up in the affected and maintained kernels can be tracked through their *hash*. Although I determined the this CVE problem with the ChangeLog of kernel 5.19, the final *patch* (kernel.org; 2022e) was applied to more kernels. *Patched* were the following kernels: 4.14, 4.19, 4.9, 5.10, 5.15, 5.19, 5.4, 6.0 (kernel.org, 2022b).

4.1.4. Going deeper about kernel.org's contribution of the problem

Generally, the *commits* can be tracked through their hashes that are mentioned in the related mails of the discussions, ChangeLogs, and so on. Also, tracking can be done if a clone of a Linux repository is available by using *git* to see or search in the *commit messages*. With regards to the elaboration of *git*, there are two "top level" *trees* of Linux: *mainline* and *stable*. For this research, I have cloned *stable* (containing all 5 era kernels) from git.kernel.org on 30.12.2022 with

```bash
git clone git://git.kernel.org/pub/scm/linux/kernel/git/stable/linux.git
```

It has to be remembered that after future *commits* and *merges*, the existing *commits*/*merges* will not change, and *git* enables us to always return to prior versions of the repository and its branches (e.g., to 5.19.16). Therefore, later clones of the repository will produce the same results. It is to be seen later if *mainline* needs to be cloned, too.

From within the then cloned "linux" folder, I switched to the *git* branch of *5.19.y* at the stage (=the very *commit*) that is tagged with *5.19.16* kernel, which introduced the *patch* that fixed this CVE to 5.19:

```bash
git checkout v5.19.16
```

The two *commits* that finally brought the *patch* into the 5.19 stable kernel can be identified with

```bash
git log 6022f210461fef67e6e676fd8544ca02d1bcfa7a
git log 6ae8aa5dcf0d7ada07964c8638e55d3af5896a86
```

The first relates to the *commit* that introduced the revised *patch* ("patch v2" in the mails) proposed and sent by Linus Torvalds to *git.kernel.org* (kernel.org, 2022a), the second *commit* is the one that transported the *patch* to the public 5.19 *stable* branch (ibid) in order to be part of 5.19.16 that was then to be released. The latter was done by the responsible *stable* maintainer Greg Kroah-Hartman. However, the code in the two *commits* is equal and thus, changes to the *patch* to fit 5.19.16 was obviously not necessary.

When evaluating the development of the discussion content and of the *patch* proposals, another institution becomes obvious: changes to *git* repositories are not transported through *https* or *ssh* using a *git* implementation. Instead, the mails themselves are used to both discuss the code and also automatically forward it to the related *git.kernel.org* repositories using special syntax in the mail. Indeed, this fosters the comprehensibility of the documentation and replaces the security guarantees of *https* and *ssh* (more widespread for synchronizing *git*) with those of *OpenPGP*. However, maintainers are likely to use *https* or *ssh* to synchronize their local with their maintained public repositories, but the two are one and the same element/link of the chain, not separated: for the *chain of trust* relevant are the *OpenPGP* signatures and the *sign-off* areas that are transported and accumulated.

Generally, *OpenPGP* is not more secure than *https* or *ssh*: all three are different technologies, all considered secure when deployed within an appropriate environment. However, when put in contrast to the *chain of trust*, *OpenPGP* enables a continuous *end to end* integrity from the initial developer of code throughout all developers and maintainers who review, test and *sign-off* that code, up to the *branch* of the respective kernel. Indeed, appropriate use of *OpenPGP* is

considered critical and when it comes to proper signatures, and certificate-related management and security issues, the kernel community is strict (Kernel Docs, 2022b; plus in the discussions of the CVE problem in kernel.org-related references) and less flexible and more formal than in the previously analyzed fields: the *chain* is enforced.

Therefore, on one hand, there are explicitly easy and informal possibilities to forward information and participate in discussions, but on the other hand, once it is about activities from people who took responsibility for something within the chain (e.g., for *sub-systems*), or about code contributions, it becomes formal and strict/rigid to enforce the institutions related to the *chain of trust*. If someone is not able to sign contributions (e.g., if from a country with related legal restrictions), the contribution has to be informally forwarded to a developer (trusted/preferred by the author) who has this capability and who can do that: the original author is mentioned and rewarded for his contribution in the *sign-off* area.

4.1.5. Pre-mail content evaluation: how is code pushed into this chain of trust?

The kernel.org (2022j) mailing archive documents that the first *patch* candidate was *pushed* on the mailing list as proposal. However, the currently analyzed mailings comply to the related Kernel Docs (2022d): it gets evident that during the time code is discussed, tested and accumulatively signed-off in the mailings, it is not added to the related public *branch* it is intended for: for the time being, it remains a *merge request*/*pull request*.

Indeed, adding it to any *branch* or repository (even if it is a *branch*/repository that is not used to derive "production" kernels from) could lead one of the many people *pulling* from the many Linux *repositories*/*branches* to accidentally *pull* the *patch* without knowing about it and thus, without considering it. Technically, this does formally not break the *chain of trust*, but it can have negative impacts at one of the many related people: it is considered/assumed that people can make mistakes, and the possibility of this mistake is explicitly excluded.

Therefore, the kernel community excludes an attack vector that would have been exploitable by regularly sending malicious code, and hoping that anyone globally *pulls* and calls it before it gets rejected. Indeed, kernel.org has a good reputation, and someone who just "plays" with the kernel without being actively involved might not know about the issue.

Instead, before a related maintainer accepts and *pulls* the content, the *trees* are not affected. The contents of the mails have already to follow a predetermined syntax (ibid) so that code and messages of the mailings can be automatically processed. Therefore, the *git* repositories get their code immediately from the mails, but everything is automated: maintainers only release it within their respective stage.

Furthermore, contributors that *sign-off* are obligated to use their real names: pseudonyms are not allowed in signatures. Yet, it can be questioned if false names are identified immediately. However, many developers are passed on the way to the released kernel, and in the further course of the stages, respective maintainers become more and more well-known. The maintainers on the lists are all well-known through conventions, they are listed as Linux developers by their companies (in this case, IBM, Oracle, Linux Foundation, SuSE) and on other websites, having held lectures on conferences or at universities, and being interconnected among each other and with companies not just by the mailing lists: the *reputation* for higher level maintenance responsibility is unlikely to be achieved using a false name or only through mailing list messages.

Therefore, anonymity is not realistic for, e.g., a *sub-system* or *branch* maintainer. However, this project is not capable to prove this assumption, and exceptions especially for organizations with huge resources to fake an identity to develop a "fake identity" towards becoming a maintainer cannot be excluded, although one case of this "fake identity" possibility does not bypass the *chain of trust*, and it would have devastating consequences for the person appearing in "public". Thus, I consider this possibility as theoretical but not practical.

With the assumption in mind that maintainers cannot be anonymous and that they are depending on their *reputation*: this achieves a type of personal responsibility and liability (legally, socially, professionally) in an informal and flexible way, and even with a false name throughout the career, developers would risk all their achievements and their *reputation* when abusing their responsibility, and risk potentially even legal consequences when acting intentionally against the interests of public good.

This relates to my past thesis, where I already identified *reputation* as a major institution in the socio-technical security provision in anarchy (Klooz, 2021, p.20). At the same time, *redundancy* as institution (Klooz, 2021, p.41) not just in technical but also in the social realms further mitigates abuse: even if a single entity of the *chain of trust* abuses the responsibility, many other entities are likely to identify manipulations. This does not just include maintainers but also anyone globally who *pulls* from their repositories. Besides the mitigation of risks, this can be also interpreted as a deterrence, and also as a protection for maintainers since the mitigation makes it unattractive to, e.g., try to blackmail them for achieving manipulations.

4.1.6. The stable and mainline maintainers: single points of failure?

At this point, and with the previously analyzed ChangeLogs in mind, the question rises if the maintainers of the *mainline* and *stable* branches can act as single point of failure. However, although the deeper analyses of the mailing contents is to come next, it can already be said with a short review of the mailings up to the fixed kernels, that any *git* instance that has been passed contained more than 1 people to *sign-off* the change: at the level of the kernel 5.19.16, the final *commit* from the recent *git* instance to the final *stable* branch was signed-off by three people before added to this branch, and the kernel 5.19.16 itself was then signed with being tested by 10 people, additionally to previous tests of individual *commits*. Later research in this project will contain a count of the *sign-offs* from the finally accepted *commit* throughout its processing to all affected kernel *branches*.

Additionally, many developers and maintainers will have *pulled commits* already in earlier stages. If a maintainer of *mainline*/*stable* would impose further changes, this will be obvious to other maintainers and developers: the outcome would be no longer compatible. At the same time, many people have to review the log of *commits* to keep on par with developments: a previously unknown *commit* would be unlikely to be overseen. Furthermore, such a manipulation at a *branch* maintainer level could cause immediately, and the latest at the very next *commit*, incompatibility that makes at least *rebasing* necessary: this shall not happen at this stage, which is why maintainers and developers could not ignore this. However, it could be argued that strong trust in and well *reputation* of the *branch* maintainers might be exploited from a social perspective, making these people to have a comparably high chance to impose manipulations. Yet, the manipulations would be evident and proven in the repositories (and signed), and anyone globally could reveal them, which leads to deterrence: just one tester or reviewer is necessary.

Furthermore, after the release of a kernel, it has to undergo further standardized testing by the operating system communities and companies, which get the kernel, can verify its integrity with cryptographically secure means (the *OpenPGP* signatures) and test it with their respectively individual testing means and measures within the continuous and holistic *chain of trust*. However, if the three related operating system communities/companies mentioned by this CVE's page exploited the possibilities of a continuous *chain of trust* is to be evaluated later. The exact number of communities and companies that adopted and independently tested the respective kernel cannot be identified reliably in this project, if at all.

In sum, the maintainers of the kernel's *branches* are neither a single points of failure at their respective stage, nor are they the end of the *chain of trust* towards operating systems that can be deployed to production use cases. However, the selection of the current two maintainers of the kernel *branches* should be evaluated independently later because it will indicate the natural selection of such posts: both are *fellows* of the Linux Foundation, as listed on the website of the Linux Foundation (2022). Yet, this is not immediately related to this evaluation and shifted to later stages.

Yet, the kernel.org infrastructure leads back to the code that is *pushed* through the mailing lists and processed automatically through respective *git* instances (instances of repositories with their branches): does the infrastructure ensure the verification of all signatures before processing code or does this depend on the large but unknown number of mailing list receivers who are obligated to verify? The Kernel Docs (2022d) clearly elaborate that the *sign-off* area *tags* "sign-off" (meaning "was involved in development or was in patch delivery path / has signed-off somewhere in the chain"), "code reviews", "testings", "reported", "acknowledged/acked" (acknowledged but neither self-developed nor self-forwarded), "suggested" and "co-developed" are tracked through the *sign-off* area at the bottom, which accumulates the email *tags* that are added over time and over the many emails, each time with the mail author signing the mail in total with *OpenPGP* and including preceding mail tags from previous mails and mail authors. The legal security that goes along with especially the "sign-off" tag is considered out of scope for this project.

However, it is not explicitly regulated that the infrastructure itself does also verification when related actions are triggered. Yet, this correlates to "trusting the developers, not infrastructure", and if infrastructure was regulated to verify, this would add only one more verification to already many other existing verifications throughout the chain. Generally, it could be argued that the simple Infrastructure of kernel.org, which only deploys the very needed capabilities each in the simplest way, limits the attack vectors to simple and well understood technologies and means. Indeed, even the *HTTPS* pages of kernel.org I elaborated so far were simple and did not depend on enabled JavaScript.

The "reported" tags are less relevant for the *chain of trust* but shall improve the *reputation* of the reporters (e.g., reporters of bugs) (ibid), facilitating and fostering their future support and professional development (this is supporting my arguments of *reputation* as an institution). Also, *tags* like "tested" or "reviewed" no longer apply once changes have been made. However, the question is if this data is "just trusted" or if it is verified by developers/ maintainers. "Just trust" would introduce potential attack possibilities that would need to be identified: indeed, social habits and social trust relationships can be powerful vulnerabilities. Additionally, this is linked to the question if and when signatures are verified in general.

It is recommended that everything shall be signed and not just the obligated things (Kernel Docs, 2022b), and it is suggested to use *git*'s capability to check signatures to verify not just code but also *tags*. This is a recommendation and no requirement, which lowers the entry barrier to enter discussions and to contribute. However, given the high value of *reputation*, it can be argued that responsible maintainers and regular developers have incentives and potentially even pressures to keep implementing these recommendations: anyone *pulling* or *cloning* from that maintainer/ developer can and are encouraged to verify the signatures (thus, *redundancy* fosters these incentives as well). Dangers for rising social habits over time to not verify are tackled by the fact that the verification process can be automated within *git*.

Given the high level of qualification and competition in Linux realms, the rejection of security-relevant measures like signatures can be argued to potentially have a negative impact on a developers professional development just like disobeying related regulations, if not more. The incentives to verify are related to deterrence, especially among maintainers with related responsibilities. At least, an attacker has to assume that many entities have incentives to verify signatures, and it cannot be evaluated how many do verification or check the *chain of trust* regularly: indeed, identifying issues around verifications can itself positively affect a *reputation*.

At the last stage, the released kernels are not just *tags* in a branch (e.g., 5.10.0 is a *tag* in *mainline* and the beginning of the "5.10.y" *stable* branch, while 5.10.1 and 5.10.2 are *tags* in the "5.10.y" *stable* branch) but are also provided as *tarballs*: a *tar* archive that contains the very kernel source without *git* but that is also signed (ibid). Yet, this project is neither capable to verify if the *tarballs* are practically verified in the same manner than related *git* instances, nor if they are in widespread use when building "production" kernels: the *tarball* can be created with a *git* command and therefore, there is no dependency to use the one provided on kernel.org.

Recommendations (instead of obligations) around *OpenPGP* indicate again that the institutional environment around Linux is less formal than the international society. However, the international society and its institutions are the determinants of the differentiation between formal and informal: if assumed to be in anarchy, the Linux system could be indeed used to derive new definitions and differentiations between formal and informal. Also, in my past thesis I already found identification that the international society also increasingly develops towards informal norms, rules and practices, such as "defend forward"-like doctrines or informal reinterpretations or blurs of "sovereignty" (Klooz, 2021, pp.22-25). However, at Linux, the "informality" is explicit and intentional by design.

4.1.7. Signing and Verifying in detail

To verify the practice about the obligated *tags* in *git*, I generated a script that (within the Linux repository I cloned above) invoked `git verify-tag x` where "x" is replaced by any Linux kernel listed with `git tag` that begins with *v5*: beginning with `git verify-tag v5.0` and ending with `git verify-tag v5.19.17` and saving the results in a text file. The result is that all these kernels have been signed off by Linus Torvalds or Greg Kroah-Hartman (one way to import *OpenPGP* keys is kernel.org using *HTTPS*). However, this does not prove that all preciding stages have been signed-off, too.

Indeed, through the previously concept of *rebasing* of *commits*, signatures do no longer apply if the *commit* is no longer deployed to the exact same order among other *commits* within the same tree, or if other changes occur. Therefore, at the early stages before a *commit* ended up in a kernel.org branch or at a related maintainer, the Kernel Docs (2022b) already note that related developers *"don't bother"* (ibid) about signatures. However, since the reviews and discussions begin after that, this is lowering the entry barrier to contribution and discussions but does not decrease security: it just makes clear that the *chain of trust* does not necessarily begin at each *patch* developer's local repository. It also remains recommended to keep everything, which is considered public, on hosting services independently available to the public (such as kernel.org), complemented by signatures. This can be interpreted as "best practice" with the following reasoning:

> 1. *Should there ever be a need to perform code forensics or track code provenance, even externally maintained trees carrying PGP commit signatures will be valuable for such purposes.*
> 2. *If you ever need to re-clone your local repository (for example, after a disk failure), this lets you easily verify the repository integrity before resuming your work.*
> 3. *If someone needs to cherry-pick your commits, this allows them to quickly verify their integrity before applying them.*
>
> — *Kernel Docs*
> *2022b*

Yet, with the CVE problem in mind, it it worth to check for hash signatures of the *commits* of the related *patch*, which have transported the *patch* from entering the kernel.org infrastructure in its initial version and the revised *patch* v2, to the related kernel 5.19.16 up to the hash of the released kernel 5.19.16 itself. Verification of *commits* can be conducted with `git verify-commit <commit hash>`. Additionally to the two *commit* hashes noted above, the hash of the actual kernel, which includes the CVE *patch*, is *d235c2b1f470f012bda26844aabf26321b1c446a*. However, when checking the three *commits* in the *git* repository, the *commits* have no dedicated signature. Obviously, the original signatures are no longer valid at this point because the *commits* are usually no longer at the same place, in the same order, or even in the same *branch*: especially when it is about individual *commits* and not *merge requests*, signatures are very unlikely to remain valid. Yet, we already identified above that the *tags* of all kernels have been signed, which ensures the integrity of the kernel. Yet, it remains the question for the integrity up to the point a *commit* ends up in a *tagged* state that represents a kernel, such as 5.19.16.

Indeed, the mailing list mails that have been archived and that remain available contain the *sign-off* tags in the chronological order, but no longer signatures. Of course the archive (lore.kernel.org) is itself secured by HTTPS once information is retrieved. Additionally, the fact that the mailing lists are open for everyone makes it easy for many other pages to also archive the kernel mailing lists. Indeed, most kernel mailing lists even mention several archive pages and not just one, creating *redundancy*. However, the question is about the *chain of trust* until these mails ended up in the archives: based upon the archives, signature verification of discussed *commits*/*patches* is not possible. Yet, corresponding to "trusting developers, not infrastructure", it remains with all subscribers of the mailing lists to do verification at the very time, and once a kernel is *tagged*, previous stages are only relevant if forensic analysis will be necessary for some reason: this leads back to the "best practice" to keep all *trees* available for such cases.

Getting *patches* into the kernel.org repositories is quite easy, and can be done by everyone who is able to follow the instructions (Kernel Docs, 2022f). And from within the mailing lists, things can be discussed, adjusted and finally forwarded throughout repositories, their instances and their *branches*, each time by the related maintainer. This leads back to the fact that many people are involved in the chain, and at least those who have responsibilities around kernel.org have strong incentives to not abuse their power: this has to be set in contrast to the fact that, as elaborated before, everyone can enter any of the related mailing lists. This means that everyone can get the emails with the signatures, and everyone, especially maintainers, are encouraged to verify signatures. Verifying can be done by everybody and automated with the default configurations of *git* put forward by the Kernel Docs: given the spread of Linux, and the many interrelated companies and communities, this is a large number of people and entities. This has again a partly informal character, but enables flexibility, and through *reputation* and *redundancy*, incentives towards best practices and *deterrence* against abuse/laziness rise - complemented by the fact that several instances have to be passed with each being able to identify issues.

Yet, at least theoretically, the security could be decreased when processing problems such as the given CVE problem, or generally severe problems: besides exploitable bugs that shall be submitted first privately to the kernel security team, *patches* for severe bugs shall be directed directly to the *stable* branch (Kernel Docs, 2022d). In the case of our CVE problem, preceding communication seems to have taken place, which might indicate the kernel security team to have been involved. However, once it was decided that it is ready to be published, the *patch* was directed directly to the *stable* branch. However, additionally, it was directed to the *scsi sub-system*. Therefore, *redundancy* and massive possibilities for verification are enabled through at least two of the mailing lists. Additionally, "reputated" and "responsible" maintainers of each plus additional personally-known developers (information of all of the involved developers is available through their company's pages, conferences and so on, linkable to the email addresses they used and that are linked to their signatures) have been active and CC throughout the mailings. Therefore, several mailing lists and several explicitly involved people could verify the mails throughout, and several have clear incentives and/or obligations to verify before passing.

Primarily, this seems to be not a decrease of stages but an increased parallelization. Given the testing stages within the Linux operating systems after a kernel has been released, it can be questioned if a kernel ends up in an operating system before the mailing list discussions and *git* operations in the kernel community have finished. Complementary, such accelerated *pulls* directly into *stable* branches take place only for the very *patches* of severe and time-critical problems, and thus the *commits* are limited to this very purpose, which makes these *commits* simple and comprehensible, and on themselves not error prone.

Therefore, what ends up in the mailing list archives has already undergone multiple verifications in any case, and many people in between do *sign-off commits* (or other tag-related activities) with "their name" and keep related to the issue and the mails (creating pressures to keep verifying and on track with the issue), although the signature documentation is only consistent for those who are signed up in related mailing lists. But all who signed up can verify each stage. Additionally, code passes several *git* instances: these will be *pulled* by related developers and testers, making it even more likely that anomalies are discovered because the *trees* create another documentation that is shared among many people who *pull* and check what has been done in the meantime to check what this means for their own work.

The *patch* of this CVE problem has been introduced to 9 kernels (kernel.org, 2022b), and based upon the CVE page (Mitre, 2022a), the Fedora Community additionally introduced it in their Linux operating system to one earlier kernel: officially, the *patch* was added to 5.19.16, but Fedora already created a *patched* 5.19.15 to be able to keep using 5.19.15 for an increased period (Fedora, 2022a). The latter's role is to be analyzed along with the other external entities mentioned on the CVE page later.

The focus on individual *commits*, which individually pass their way through discussions towards one or more different kernels, confirm another institution that is documented in the Kernel Docs (2022d): *"one problem per patch"* to keep everything comprehensible and less error prone. This indicates that *merge requests* / *pull requests* (both are considered synonymous in this project), which consolidate many *commits*, are on average not intended for adding *patches*. With this indication in mind, it is even easy to explicitly identify this intention: the related Kernel Docs (2022e) page for *pull requests* only intends these for maintainers who might use that for synchronizing with each other.

When reading the content of the mailings towards the problem solving *patch*, it gets revealed that the developments during the problem solving do not consider security in a dedicated manner, but simply aim to solve a problem: as far as it concerns the problem solving, the security team that is to receive information about exploitable bugs privately (to not publish them until immediate measures to mitigate the issue in *stable* kernels have been identified) remains the sole security dedication, and it is a security dedication that only applies to the initial stage of receiving external information: the discussion around the first *patch* draft is already public. The community seems to informally assume that, with the operating systems having an immediate measure available for their "production" systems, the massive knowledge and problem solving capabilities in the community are more effective and faster than potential attackers.

However, complementary, it is related that even exploitable bugs in the kernel are in most cases hard to exploit because an attacker needs in most cases immediate possibilities to interact with the kernel of an operating system, which is in most cases not possible by default. However, the latter will always also depend on the operating system providers and the system administrators, and secure by design also implies explicit security: do not assume the system administrator to be sufficiently competent. Yet, the assumption that the kernel community is more effective than attackers, given its contained and proven capabilities (related contributions from well-known developers, companies, universities, organizations) after the bug information has been published, can be expected realistic, although it cannot be proven. Additionally, both later stages (operating system providers and system administrators) can each on their own mitigate related issues: only one of both needs to tailor in time, which mitigates single points of failure. This decreases the attraction of attacks: an attacker needs to impose massive resources to be fast in exploiting a bug, and it will only work if all three stages (if not more) are slower than the attacker, and the attacker needs to know in advance if it will be possible to access the related kernel in a way that makes the related bug exploitable. Of course there is never a 100% security, but the limited likelihood of a proper return on investment can be seen as another deterrence.

4.1.8. The problem solving environment: the infrastructure's role in the institutional environment

Because there is a type of central infrastructure at kernel.org, I checked for documentation or other indication about the role of the infrastrcture itself in ensuring security. Generally, the infrastructure fulfills all "best practice" requirements for security by enforcing secure protocols like *SSH* and *HTTPS*, and by providing valid *x.590v3* certificates over secure *SSL/TLS* channels. The infrastructure itself fulfills what is expected from an information security perspective when it comes to its security means when accessing or communicating with it. However, although this is usually considered sufficient in most cases, the institutions of *trusting the developers, not infrastructure* and *assume that any part of the infrastructure can be compromised at any time* create different circumstances (and imply different requirements) than traditional security concepts: as elaborated before, there are many redundant HTTPS-accessible archives of the mailing lists, many officially mentioned as archives on the respective mailing list sign-up pages (and within mailings). The security role of the infrastructure (and other infrastructures) in terms of providing archives seems clear. However, when it comes to the ongoing/current code development and the actual *chain of trust*, the role of the infrastructure when it comes to security was already partly indicated before.

Indeed, the kernel.org infrastructure seems to not be included in the *chain or trust* in terms of verifying *OpenPGP* signatures: *trust developers, not infrastructure*. The infrastructure has to be assumed not trustworthy by the developers. However, as previously elaborated, a large bunch of people has the possibility to verify, and many have clear incentives, some standing with their name for what they pass and/or *sign-off*. The different repositories stored on the kernel.org infrastructure are not controlled centrally but by their maintainers. They *push* and *pull* changes with their local *git*.

The public repositories are likely to be used to *distribute* repositories to new people who rely on its integrity for the moment they get their initial forks/clones, but many existing developers also *pull* from these public repositories and would get the information of manipulations immediately. Since their job makes it necessary to know about changes, at this place they practically need to check what is added/removed, and this takes place through multiple instances before it ends up in an actual kernel. Additionally, it is another institution that was also already found above in the Kernel Docs: kernel.org repositories, and all other repositories, shall be are also mirrored by their maintainers on multiple other hosted *git* instances. E.g., Linux Torvalds mirrors the *mainline* in at least GitHub (2022).

Additionally, the major *branches* of kernels can be found many times on different *git* hosting services from multiple people. Beyond the points about the infrastructure and verification that I already identified before, there is another point that is indicated at this section: that the infrastructure does not do verification, and that the infrastructure has thus to be assumed not trustworthy, mitigates that developers over time fall prone to "intuitive" social habits and get used to "just skip" the verification with the "intuitive" assumption that the likelihood of the infrastructure being manipulated at the very moment is very low: from a "best practice" point of view, not doing verification can be argued to be equal to manipulation. So the incentives for "coziness" social habits is much lower.

The increased simplicity of the infrastructure, identified earlier, and the absence of verification on the infrastructure level can thus be seen as security improvements in the given *trust developers, not infrastructure* environment.

A less holistic security approach could imply that more verification is always more secure than less. However, besides the "simplicity of infrastructure" advantage, the additional code in the infrastructure would be likely not as widely deployed, tested and reviewed for this very use case as the kernel or generic *web server*/*git* implementations that do not contain adjustments, which would make the additional code error prone more than other parts of the infrastructure: if kernel.org does not deploy complex JavaScripts or other *callable*/*executable* technologies, the code that is deployed can be generic and deployed with default configurations without adjustments, whereas the content of the pages does itself not have exploitable agency (text cannot be *called*/*executed*). More complex technologies usually need more complex adjustments in their configuration that can change their agency and thus, accidentally create an outcome that does no longer correspond to the generic/default testings of the technology (at the worst, introducing attack vectors).

Indeed, the identified manifest of the *trusting the developers, not infrastructure* and *assume that any part of the infrastructure can be compromised at any time* institutions reveals a major difference to most security approaches that naturally develop single points of failure: a centralized infrastructure is considered itself a point of failure and shall be avoided explicitly: do not allow incentives that can make people falling prone to assuming the infrastructure being trusted.

Traditional approaches tend to focus on central points of responsibility and service provision. This also counts for *distributed* services that have a centralized management, while management often implies two single points of failure: an administrator (social) and one or more machines that can be used to manage/manipulate everything (technical). However, the advantage of the Linux architecture at this point has a limitation: related people have to be correspondingly qualified.

From a holistic socio-technical point of view, it can be considered a strength that the kernel.org infrastructure is not included in the *chain of trust*: Yet, the maintainers and developers are people, and people can get used to each other and develop trust into each other. Therefore, exploitable habits cannot be excluded.

4.1.9. Indicative excursus to Minnesota: related or comparable to this CVE problem? A breach?

When elaborating the above and evaluating existing information related to the mentioned issues and intentions of the kernel community, the correlations to incidents around the University of Minnesota in 2020 and 2021 can hardly be overseen. They are not explicitly and immediately linked to the CVE problem, which makes them out of scope for later summaries and conclusions that relate to this CVE problem. However, the comparabilities and the explicit tests for attacking the infrastructure, which I currently evaluate, makes them a complement whose attack vectors and outcomes should be set in contrast and comparison to the CVE problem research, while evaluating how far there are parallels with the currently analyzed CVE problem. Given that these incidents are the only practical cases of their kind, it might shed a different light on my recent and next analyses and derivations, which might increase the capabilities of this project to interpret its data.

The *Report on University of Minnesota Breach-of-Trust Incident* from the Linux Foundation's Technical Advisory Board (kernel.org, 2021) sums up this case and contains all relevant links to discussions and communication around the case (including external/independent data/links), but also links to the actual mailings in which attempts to introduce malicious code into the kernel have taken place. An elaboration in the mailing list archives about the case ends up at the same pages noted in the report. Therefore, the report will be noted as single source, which will also include data from the pages that are linked within this report.

A research group of the University of Minnesota has tried in 2020 to introduce malicious code into the kernel through faked mail accounts. Their own publication on GitHub (linked in the report) claims that they revealed weaknesses in Open Source communities. However, at least with regards to the kernel, the attack did not work out: 4 of the 5 *patches* had been rejected, and the one that has been introduced to the kernel did not contain a vulnerability. The latter fact has been discussed in mailings and in the report, and it was assumed that it was an accident that there was no vulnerability contained. The assumed reason was that the authors did not understand the underlying architecture so that the code that was aimed to have a vulnerability, was "accidentally" absolutely secure. Indeed, the latter's related *commit* was later removed only to enforce the decision to remove any contribution from this university, which was decided several months later, after the later incidents in 2021.

However, this proves that even a name that sounds real and that has an *OpenPGP* signature is easy to fake. Yet, the case has shown that the developers of the kernel are aware of that fact and do not "just trust" *patches* and signatures from unknown people: sufficient review and testing seem to take place.

Yet, the later 2021 incidents might be better cases to challenge the architecture's security: a member of the same research group of the University of Minnesota started to submit several *patches* to be introduced to the kernel. However, he was not using fake mail addresses but his official email address from the University of Minnesota with "@umn.edu". His later reasoning might be not relevant for this project, but it might be considered that he made it to introduce 3 *patches* that contained vulnerabilities, although it remains unproven if these vulnerabilities were intentional. In discussions, it was partly questioned if this revealed security issues around the kernel. However, if the two group of incidents are compared to each other, it can be said that only *patches* that were officially submitted by a person with a name that could be linked to "@umn.edu" could pass their way to the kernel.

From a purely technical perspective, this can be seen as vulnerability. However, from a socio-technical point of view, it can be said that, although this cannot be proven with the existing data, the *chain of trust* has worked: what has passed could be linked to a trusted institution and real people, who had to take responsibility for what happened, and thus, who had no incentives to exploit this for criminal or comparable reasons. Therefore, a real attack and that these entities exploit the outcomes remains unlikely: if it was a real attack, they would have themselves documented noteworthy evidence against themselves, and in all cases, many people were formally obligated to review and pass their contribution, making it realistic to get caught. It is unlikely that sophisticated hostile attackers would have taken that risk ("deterrence").

The research was questioned by many entities and researchers, and was also questioned internationally in its quality (not just the submitted code). If properly planned, prepared and conducted, it can be questioned if that project had taken place, given the unknown outcome for the researchers' *reputations*, up to the possibilities of prosecution.

Yet, many developers concluded that in future, it should be put more attention to better review even trivial *patches*, and not just trust in addresses from known universities. However, from a socio-technical point of view, the institutional environment in which the *patch* reviews and testings take place seems to remain unchanged. And even if developers are now for some time imprinted to be more careful with comparable cases, it is realistic that they develop towards the same old habits if the institutional environment does not change. However, given the limited analyses and the often informal institutions around the Linux socio-technical architecture, it remains impossible for this project to evidently determine if the informal elements of the institutional environment remain exactly the same or not. Yet, the possibility of the latter leaves the question if, in the foreseeable future, a socio-technical vulnerability can be exploited by attackers that capture, e.g., university email addresses from known researchers to exploit these addresses without the owners being aware. Nevertheless, this project will not be able to indicate or prove if that will become realistic.

Additionally, there are more stages for review and testing after the released kernel: indeed, all the operating systems that deploy a Linux kernel have their own measures and standards to do security- and other types of testing and review. It remains unknown if the three vulnerable *patches* would have made it to "production" deployments: many tests (which might include further reviews) take place after the kernel release, and all the different tests include different environments and different means which introduce different perspectives and different coverage.

Further, as indicated before: the general institutional environment and *tags* like "reported by" create incentives for every developer, engineer and penetration tester to try to find bugs/flaws to improve their *reputation*: before a kernel is used for critical "enterprise" use cases, it has to pass a much longer way, and it can take in some cases years after a *mainline* kernel's release (e.g, 4.14) before it is deployed in such critical use cases (this refers to "Long Term Supported" kernels that develop up to, e.g., 4.14.309). Thus, an attacker has to expect the vulnerability to become revealed before, while it would also remain unclear if the vulnerability will be still exploitable at the time of introduction to "enterprise" use cases: this limits both the likelihood of success and the potential return on investment for such attacks.

Although the involvements, interactions and interrelations with the Linux Foundation's *Technical Advisory Board* indicates more relevant institutions, this remains out of scope as long as the determined problems do not involve it.

In the end, this could not evidently disprove the *chain of trust* and the other related institutions, and from a socio-technical point of view, it can be argued that the *chain* was at least partly even confirmed. Nevertheless, just like with *SHA1*, it can be argued that the kernel community is deadlocked in its institutional environment, and although it is not proven that it needs adjustments, the lack of interest in and legitimization of reflections on and analysis of unforeseen occurrences could indicate inflexibility about adjustments and tailoring when indication occurs.

Yet, this argument remains probabilistic as long as it remains probabilistic if adjustments are necessary: neither *SHA1* nor the Minnesota cases have proven so far that adjustments are necessary. Based upon what we know, *SHA1* is still sufficiently secure for this use case, and none of the Minnesota cases ended up in exploitable use cases. It remains to be seen if the kernel community proves sufficiently flexible to adjust in a timely manner if any security-critical issue will need immediate response: at this point, it is not clear if the legitimization of critique of the established Linux institutions is limited and thus, if it will be capable of adjusting early (=before an exploitation takes place) if it is really necessary.

4.1.10. The kernel and Fedora, Debian and OpenWall

It is probabilistic if the communities and organizations of and around Fedora, Debian and OpenWall are independent architectures or part of a bigger Linux architecture. However, based upon the goals, definitions and determinations of this project, they remain considered part of the socio-technical Linux architecture: any kernel that is deployed in

"production" has passed at least one of these communities or organizations. Additionally, these communities and organizations provide much of the testing and review, while they are also providing *feedback loops* to the kernel community and contribute directly and indirectly to the kernel's development. Members are overlapping.

Concerning the current CVE problem, the MITRE page (2022a) relates to contributions of the three mentioned communities in processing the CVE problem, starting with Fedora. Generally, the three links that are named "FEDORA:FEDORA-2022-1a5b125ac6", "FEDORA:FEDORA-2022-2cfbe17910" and "FEDORA:FEDORA-2022-b948fc3cfb" relate to the pages of the *patched* kernels in Fedora's update system (Fedora, 2022a; 2022b; 2022c). These are "usable" kernels created by Fedora's *build system* from the kernel's source code, based upon Fedora's configuration tailored to Fedora's operating system: three kernels for the two currently-supported Fedora releases, and one for the development version. Fedora's mailing lists contain further information of what changes these kernels include (Fedora, 2022f; 2022g; 2022h):

Interesting is the *distribution* of the *redundant* information: every page that is linked contains itself all related links (which also leads to recursions), whereas overarching points are "redundant" throughout pages/links. This itself could be used to define/delimit the Linux architecture and what belongs to it: define/delimit the socio-technical Linux architecture by the provision of related information that is integrated by any link at any contained page, starting from the inner *git*-defined kernel community. So far this is a different approach to define/delimit this architecture compared to my delimitations/definitions, but the outcome (the architecture that is delimited/defined this way) can be argued to be equal to my delimitations/definitions, although this remains probabilistic. However, the information-based definition/delimitation is more comprehensible and less interpretable than my approach, and its correlations/ analogousnesses (in the outcome) to my security-based definitions/delimitations, with which I work in this project, illustrate that information flow links to security.

lwn has been already a source several times before, aggregating information from kernel.org by publishing automatically or manually what is added to specific kernel.org mailing lists, and to aggregate information that is explicitly forwarded to *lwn* by its own mailing address that can be set CC. The mailings about Fedora's kernel releases are put on *lwn*, too (Fedora 2022i; 2022j; 2022k). With this type of accessibility in mind, the *redundancy* among different type of sources, with many sources having a clear scope of what they contain and what not, it can be argued that the way of *redundancy* and way of *distribution* in the Linux architecture forms a type of intuitive *NoSQL* database. Just like in, e.g., Apache Cassandra, *redundancy* is not seen as disadvantage for database designs/structures because the immediate availability of related data for a specific "query" is the major goal that is to be achieved: e.g., if it is about data around released production-ready kernels and their interrelations with issues that affect their production deployment - without limiting the data to kernel.org or any specific Linux operating system but with limiting to kernel-specific topics without other operating system topics, then *lwn* is a proper point to start searching on in order to get only but all the very related information that is required by the "query".

In the overall architecture, this massive and effective availability and accessibility of required information can be determined as another institution: a "NoSQL-like" institution, but it presents itself as a database only from a holistic socio-technical point of view. Yet, this is again something informal, which also means that this institution's advantages are primarily usable by people that are used to this architecture, although this does not mean that it is limited to developers.

Additionally, beyond the likely but in this project not provable possibility that the mailing lists are highly intertwined, the maintainer of Fedora's kernel is one of the testers of the kernel at the stage of kernel.org (kernel.org, 2022a; Fedora, 2022a; 2022b; 2022c; 2022f; 2022g; 2022h), which generally supports the assumptions about interrelations and the blurred affiliations among people and entities.

However, at this point, I became aware of another limitation: I have been myself tester of two of these three Fedora kernels (Fedora, 2022b; 2022c). Yet, as documented on the two pages (Fedora, 2022b; 2022c), I was not involved in any testing related to the CVEs: I was just testing if the kernels boot and work properly in average tasks on given hardware,

and I ran the *regression test suite* (a tool that can be cloned from Fedora's git instance: Fedora, 2022d; 2022e) for both kernels, which contains automated standardized tests that shall run for each kernel on different systems/hardware of different entities to see if all have the expected (and thus, same) results. Both had no relation to the CVE. The *regression test suite* is a general tool, at the time of the project unchanged for at least two years (Fedora, 2022e), and not tailored to a specific kernel or problem: practically, it is a tool that is to be started after booting a kernel to then report if the tests pass, or if any fails, report the logs. Thus, I had no relation to this CVE's processing.

Yet, all information around Fedora as noted on the Mitre's CVE page is related to test, release and introducing the *patched* kernel, but not to the initial solving of the problem up to the *patch*. Yet, the "bugtracker" of Fedora has an open issue for the bug, which collects links to different mailing lists and tracker related to the problem (Fedora, 2022l). Again, this creates *redundancy* and ensures that everyone who ends up at one place gets the links to all related information, while there are multiple access vectors to the information, linking potential discussions that might also not have all the same focus. Also, following the "NoSQL" institution, the different "entry points" to the issue might focus different types of information and links. The ticket in the "bugtracker" of Fedora is focused on making aware and follow the development in the initial elaboration of the issue. At Fedora's "bugtracker", the reporter was primarily to make aware of an issue already reported somewhere else, and to start evaluating how far and which Fedora releases are affected, and how to respond immediately (ibid).

Additionally, these Fedora pages are linked to another Fedora-specific ticket that is about this CVE vulnerability in general (Fedora, 2022m). This ticket (ibid) seems to have not been involved in the actual problem solving but tracks the related developments as far as they concern Fedora. Thus, Fedora itself seems to have not been involved in the solving of this very problem. Yet, this shows that *redundancy* does also provide many places for inputs to the Linux architecture: from the kernel down to the operating systems, bugs and issues can be reported anywhere, with respectively responsible people to *distribute* information through ubiquitous *feedback loops*.

Indeed, at the Fedora "bugtracker", everyone can create a free account to file issues. This also illustrates again the informal character: there is no explicit regulation about where to file. Yet, most provided documentations suggest to file issues against the operating system so that its developers can verify first if an issue is operating system-specific or if the issue needs to be forwarded to, e.g., the kernel. Nevertheless, this remains informal and during the problem determination (see related links), it became already obvious that regular contributors tend to directly file against the causing element, which seems to be accepted if they seem to know how to identify what is the cause. Indeed, the interrelated and blurred entities make it impossible to clearly say when someone has something that has to be reported against an operating system, and when it is related to immediate, e.g., kernel development/testing. Thus, the "do file against operating system" argument can be interpreted to be directed primarily to average users to avoid them "flooding", e.g., kernel mailing lists. A simplified explanation of this "institutional agency" could be: "report to your operating system, and the more experienced you become and the deeper you get used to the structures and informal norms of the overall Linux architecture and its interrelations, the more you can go beyond and report to the causes, such as the kernel, if you know how to identify it explicitly". Yet, this "institutional agency" is probabilistic and cannot be proven or fully analyzed within this project because this this would be a project on itself.

Finally, it might be noted that the pages around Fedora/Bugtracker, as far as related, also fulfill current standards in terms of security and cryptography: all negotiate preferably TLS 1.3 and provide secure x.509v3 certificates. Yet, much more is focused on using user accounts to login, using usual but secure means including 2FA, while the mailing lists do not enforce signatures formally or informally. However, it might be noted that the mailings seem to be neither intended nor used to exchange code. Additionally, this shows that mailing lists of operating systems might be much easier to be accessed and used for users who are not sufficiently experienced to work with the kernel mailing lists and their requirements. Still, the *feedback loops* can transport relevant information up to the kernel but also filter what is not relevant.

However, beyond the already evaluated pages, the Fedora tickets also link to the Openwall mailings that are also noted on Mitre's CVE page: OpenWall, 2022c; 2022d. Indeed, OpenWall seems to be a Linux operating system to which the original reporter "hdthky" is related to: the reporter opened the related topic on the mailing list (2022a) after the first version of the *patch* has been put to the related Linux mailing list I analyzed earlier. As on Fedora, the corresponding link was provided. Additionally, the reporter provides some more information about the bug and made clear that it was discovered by a two-person team including him from "IceSword Lab" (ibid). OpenWall (2022b) introduces the CVE number to the topic after it was assigned, making it related to the topic in general and obviously for search engine indexing. It would be realistic but not provable that the information provided by the reporter at Openwall equals what was provided to the kernel developers. Nevertheless, using the reporters post, it can be easily found out that it was spread also to other related mailing lists.

Concerning the infrastructure, OpenWall fulfills the updated TLS 1.2 standard and provides a valid x.509v3 certificate, being secure in these respects. The mailing lists are comparable to Fedora, not enforcing signatures. Although the reporter provides the vulnerable code, there is no indication that this mailing list is intended to be used to transport or evaluate code. In this case, the vulnerable code was provided obviously for information purposes, with a link (not the code) to the related Linux mailing list for the correcting *patch*.

As far as it concerns the Debian link on Mitre's page, it is only a security update that includes information about CVE-related problems that had been fixed in the then recent Debian kernel 4.19.269-1: this included the analyzed CVE-2022-40768. Therefore, more analyses are out of scope for this project.

Yet, this links to another implicit limitation that has been already previously indicated: there are a lot of companies and communities related to the Linux kernel, its development/testing and maintenance, and the development/testing and maintenance of Linux-based operating systems: in the end, many of the related entities are out of scope for this project.

4.1.11. Avoid one selection to make the analyses

However, as elaborated earlier, CVEs are not in focus in Linux. Therefore, it could be corrupting to rely on the selections of CVE/Mitre. Yet, the *commit* hash values and the CVE number, which - as indicated in previously analyzed pages/mailing lists - tend to be mentioned/linked in page/mailing list topics and elsewhere, including the strong public and widely *distributed* information related to handling the problem, and offer effective possibilities to identify if more is related to the problem's solving. I have analyzed the following search queries using google.com and duckduckgo.com:

"CVE-2022-40768" "Linux" "stex" → the term *stex* ensures that pages without technical information at all are ignored

"scsi: stex: properly zero out the passthrough command structure" → this is the major topic in the kernel.org mailing list, which tends to be referred to on related discussions

"https://lore.kernel.org/all/20220908145154.2284098-1-gregkh@linuxfoundation.org/" → link to the initial mailing list mail that contains the first *patch*, which is referred to on related discussions along with or instead of the link to the final *patch* "v2"

"https://lore.kernel.org/all/YxrjN3OOw2HHl9tx@kroah.com/" → link to the mailing list mail that contains the second and final *patch*, which is referred to on related discussions along with or instead of the link to the initial *patch*

In each case, I assumed relevant hits to be on the first two pages, not analyzing page three and later. Yet, besides the already elaborated pages and mailing lists, the majority of hits gave further indication of the strong *redundancy* and mirroring of information and mailing lists/mails. There is no indication that further companies and communities have been involved than those already noted above. Nevertheless, several Linux-related organizations provide information

about which of their Linux operating systems (which releases, which kernel/operating system versions, and so on) are affected: SuSE (about SuSE Linux Enterprise and openSuSE), Red Hat (Red Hat Enterprise Linux), Ubuntu (Ubuntu, Debian), Oracle (Oracle Linux), Amazon (Amazon Linux).

4.1.12. People to create compatibility between redundancy and consistency

Interesting is the compatibility between *redundancy* and consistency despite the streams of information being each accessible individually: so far, there is no indication for inconsistency (e.g, of discussions) in all these masses of *redundancy*. The overall architecture seems to be able to informally keep this condition in a stable manner, while individual slips are compensated without allowing inconsistent developments: the people-centric and decentralized architecture seems to exploit people's capacity to balance issues intuitively, which machines cannot do, and the "community" seems to be capable to keep and enforce this "discipline" in its public spaces. On one hand, this capacity seems to foster (if not facilitate) the massive flow and exchange of information in an sufficiently efficient and comprehensible manner. On the other hand, there is not yet indication for it creating vulnerabilities in the Linux architecture.

Contrary, the same capacity is in traditional realms the origin of vulnerabilities: average employees in traditional architectures intuitively solve problems and accidentally create vulnerabilities that way: e.g., if a new tool does not work with the pre-installed browser, so employees manage to put portable browsers on their workplaces, which can then remain without security updates for years. At the same time, they feel to be able to manage the problem and do not report it, keeping hidden both the tool that is not working natively but also the unupdated portable browser. The employee case creates itself an unintended input to the organization that if it remains uncompensated enables further, maybe hostile, unintened input. In the Linux case, on one hand, unintended input can easily appear given that anyone can contribute, but at the same time, the community is correspondingly used to compensate, and can do that easily given that the information of the unintended input is public. Indeed, the context can make one and the same thing an advantage and disadvantage.

A clear difference lies in the security by design: in one case (Linux), each individual can be balanced and there is no single point of failure. If individuals fall prone to incentives that rewards behavior that is disadvantageous for the majority (this can be intentionally but also unintentionally), the majority can respond and compensate. In the other case (the employee in the portable browser example), the employee can act as single point of failure and create issues that can remain hidden forever.

However, the outcome of this capacity in this context is a condition, but itself not an institution. Yet, it illustrates the agency facilitated by the related institutions in the related context(s), and might be worth to be analyzed on its own to capture further indications: secure by design is not sufficient to fully capture and elaborate the related social dynamics and behaviors. However, although being relevant for this project, dedicated analyses of this capacity of people in different realms would be a project on itself that would have to start with collecting and assessing related bibliography. However, the condition and the related agency will be included in following analyses and derivations. Indeed, it cannot be excluded at this point that it represents (directly or indirectly) a major element in the security of the Linux architecture.

4.1.13. Steering Linux: the mainline and stable branch maintainers

As proposed earlier, the *mainline* and *stable* branch maintainers' steering role in the discussions analyzed so far makes it worth to have a dedicated look on their own natural selection and role. Given the person-centric architecture, this should not contain a narrow focus limited to the "position" that is to be filled by an individual, but consider the individuals who "make" the respective post: in this architecture, it is not yet proven if the person makes the role or vice versa. Nothing should be taken for granted at this time. Indeed, there are no formal definitions or determined responsibilities that go beyond "maintenance" of the respective element (*branch* or *sub-system*). This makes the precise role dependent on and shaped by the respective person that is holding the "position" and this person's dynamics with the remaining community.

In terms of Linus Torvalds, the maintainer of the *mainline* branch, he is the original inventor of Linux and dominating the *mainline* development from its beginning (and Linux development in general from its beginning). Therefore, his role has not evolved with the Linux Foundation or another organization but has already existed before. Yet, given the kernel 5 era period this project is focused on and limited to, periods before the Linux Foundation are out of scope and are expected to blur results about the kernel 5 era socio-technical Linux architecture. The same counts for the role of the *stable* branch maintainer, Greg Kroah-Hartman, who had been holding his role throughout the kernel 5 era.

In both cases, it can be said that the kernel community informally tailors to whom it wants. The Linux Foundation has no authority over them: code is owned by the respective developers and published under the GPLv2-only license (kernel.org, 2023a). The trademark Linux is considered out of scope since it is only a name that is not related to the code and its legally possible use cases. In the end, the Linux Foundation can only choose whom it supports. Yet, the *mainline* and *stable* maintainers are both employed and sponsored by the Linux Foundation, and who is to be supported is decided by the Linux Foundation's members.

Because the members of the Linux Foundation make up the major contributing companies of Linux (Linux Foundation, 2023b), whereas these companies consolidate much of their efforts in the Linux Foundation (which excludes these companies' own developers who also make up a large amount of contribution), it can be still assumed (but not proven in this project) that the Linux Foundation has noteworthy capacities to support individuals or infrastructures around Linux and thus, to create strong incentives.

The Linux Foundation is to create a *"neutral home"* to coordinate and support Linux and other related projects and their "ecosystem" (Linux Foundation, 2023a). The membership is paid with different levels of members, while the level depends on the payment. However, dedicated analyses is currently not possible because the Linux Foundation does no longer provide explicit information about the exact payments, obligations and rights of the different member levels, or how they impact the different boards.

There are still pages that contain the information that used to be published also on the website of the Linux Foundation, but the latter removed this information from their pages and there is no possibility to verify if the information from other pages is still up to date and applicable to the just finished kernel 5 era. There are some sub pages of the Linux Foundation that still contain some information about the membership levels, but these pages are indicated to be old and not updated for long (e.g., these pages contained links that do no longer exist or that are forwarded to other links, which do not contain the type of information proposed). Thus, I assume these pages to refer to the previously published information so that I cannot verify if this is still up to date information. This is not appropriate to be included in analyses, derivations and conclusions of this project and might be corrupting results. Therefore, this analyses has to be interrupted at this point.

Yet, the *mainline* and *stable* maintainers are along with others sponsored "Fellows" of the Linux Foundation to work full time on the kernel (Linux Foundation, 2023c). Also, we know that the Linux Foundation has different types of corporate members who make up its agency. Therefore, having a look on the members makes sense and can be set in contrast to the known entities I already identified in previous analyses.

The highest level of membership is Platinum, which includes 14 companies (Linux Foundation, 2023b). From the companies already appeared earlier, 3 are included: Red Hat, Oracle, Microsoft. Additionally, there are 17 Gold members, the next highest level: only one company that appeared in previous analyses is contained in the Gold list: Google. Additionally, there are 1233 Silver members, which shall not be considered based upon the assumption that their impact is limited compared to dominating Gold and Platinum members: the information from the likely-out-of-date pages about membership level rights and obligations 2 paragraphs above indicate that Silver members had no powers to shape the leadership and advisory boards (and thus, Fellows) at the time these pages were set up. The project assumes that this has not changed, which is a limitation of the project given the lack of information.

At this point, it might be noted that there are only 8 Fellows. Only three of them have appeared on any of the mailing lists and analyses: Greg Kroah-Hartman, Linus Torvalds, and Shuah Khan. Shuah Khan is a maintainer of several *sub-systems* and was involved in testing the kernel that fixed the CVE problem (kernel.org, 2022a), having added a "tested by" tag to the 5.19.16 kernel's changelog.

Further, the previously mentioned *Linux Technical Advisory Board* contains 9 members, among them Greg Kroah-Hartman, Kees Cook who was involved in the evaluation of the previously analyzed "Minnesota case" (Cook, 2021), Jonathan Corbet (who is the editor of *lwn*, elaborated earlier), Ted Ts'o (a kernel developer and a maintainer of several *sub-systems* who was involved in the previous mailing list discussions), and other members from companies that are Gold or Platinum members.

Although the analyses of the Linux Foundation cannot be deeply elaborated for previously elaborated reasons, it is worth to note that there is no clear definition of when someone is sponsored from and based at the Linux Foundation, when someone works at one of the Linux Foundation's members, and when people that do not even belong to a Linux Foundation member are put into a board. The advisory board contains trusted members that are involved and reputated in the community, as indicated in the mailing lists and articles previously elaborated, being acceptable also to the Linux Foundation members who somehow appoint them, while the community so far accepts them.

Yet, the institution of the "Fellows" remains blurred: these people are formally "neutral" Fellows, and it is the community who makes them to, e.g., *stable* maintainer: *stable* maintainer is who is treated by the majority of the community as such. This is formally not enforceable because code remains owned by the contributors (no individual entity dominates the development → no single point of failure), whereas it is not formally regulated but the outcome of dynamics (out of scope of this project) if people are first "Fellows" to then become maintainer or vice versa: Formally, Fellow and maintainer are two different positions.

This proves that the Linux Foundation can create incentives, but not enforce its will. However, the contributing people/organizations and the Linux Foundation members are overlapping. Correspondingly, it remains that there is no formal regulation, at least not publicly available, what the Fellow roles have to fulfill: they are supported, and the Linux Foundation members seem to assume that the *reputation* and the dynamics of the community ensure the rest.

First, the CVs of the Fellows (Linux Foundation, 2023c) make clear that they are not all related to the Linux kernel, but also to other Linux Foundation projects. Only 4 are related to contributions to the Linux kernel, whereas the 2 of these 4 that have not yet been appeared in the project are not in maintenance or comparably responsible roles.

It remains indicated that the decision of who becomes a Linux Foundation "Fellow" and who is employed by another (member) company/organization is not clearly defined and seems to develop informally based upon the needs of the socio-technical dynamics.

A phenomenon whose origin is out of scope and that cannot be evaluated in detail within this project, although it is probably a shaping determinant of the Linux architecture, is the fact that the person-centric architecture also seems to shift the "constants": not companies are the "constants" (e.g., in maintaining a *sub-system* or fulfilling a specific task/role), but people and potentially their networks/communities. Indeed, when reviewing the CVs of developers, it seems that changing their employers is a regular phenomenon while their own role is the "constant". With this in mind, it could be possible that not just their role but also their community (or, communities) are "constants", with the sponsoring companies being the "variables" that are interchangeable. When reviewing other critical components of modern architectures (e.g., *systemd*, which is the initial process that manages and shapes the agency of a Linux kernel at most Linux operating systems), these components' constants are their maintainers (= individuals) who, based upon the previous results, depend more on their *reputation* than on their employer's will (likely being able to replace their employer if the *reputation* remains appropriate), while the companies that are involved in maintenance are more likely to change.

At the same time, another institution is indicated at this point and becomes apparent, in conjunction with previous elaborations: it is assumed that people who contribute take responsibility for their contributions and keep maintaining them, and when it comes to major components, they are assumed to actively search replacements before they cease their contribution. Yet, these are informal dynamics that are linked to *reputation*, but are themselves a *hand over before resign* institution.

Precise characteristics of the behaviors that rise in the Linux architecture, about how people behave against each other and how their dynamics evolve with each other, is out of scope for this project, although it could add valuable perspectives on and indications about the architecture. Indeed, the criticisms that can be observed in strategic discussions and articles of and around the kernel (not only with regards to its development) about behavior in the mailing lists and related platforms that can be offensive or insulting, are worth dedicated analyses in future projects. Yet, the project has not yet covered contents that indicate related issues. However, when analyzing such issues, these could be related to intercultural issues because the Linux community is by nature highly diverse in many respects. But this shall be evaluated by future projects.

Generally, all together manifests another institution that at first glance loosely reminds on a technocracy: people do not insist on contributing to every decision, as formally and informally practiced in what people tend to perceive as modern democracies. Instead, people focus on contributing to decisions they have knowledge about, but neglect or even ignore other decisions and other discussions. However, the architecture, its institutional dynamics and the formal-informal shapes should not be simplified to traditional governance types like technocracy, which has at the best a few superficial correlations in its outcome in common with the Linux architecture.

4.2. CVE-2022-3435

4.2.1. Getting into the next problem

The next problem, CVE-2022-3435, is comparable to the first problem: it is a time-critical and security-related problem, being linked to a CVE. To avoid repetition, I will compare the second problem against the first instead of doing a complete elaboration: the goal is to identify what both problem solving processes have in common (which indicates if these two problems' institutions are common to the Linux architecture when solving such problems) and what can differ in time-critical security-related problem solving. Obviously, what differs in problem solving processes also offers indication for the Linux architecture.

CVE-2022-3435 is related to a function related to IPv4: "IPv4 handler" (Mitre, 2022b). The flaw allows an attacker to remotely read specific data from the memory without authorization ("out-of-bounds read") (ibid). Concerning the CVE number assignment, this problem illustrates that the Linux architecture does not focus on or wait for CVEs: the CVE was assigned on 8th October 2022 (NIST, 2022b), whereas the first *patch* draft was already in discussion on 5th October 2022 (kernel.org, 2022o). Just like in the first problem, the public discussions begin with publishing the first *patch* draft. Therefore, it can be only assumed that the reporter has started with informing the kernel security team.

However, it is evident that the *patch* draft mailing (which marks the beginning of the discussion) is not the start of the problem solving: the person who started the discussion with the first *patch* draft is not the person who reported the flaw, which is documented in the first mailing with the "reported-by" tag (ibid). The reporter is affiliated only with a private email address, not bound to any company. Concerning the problem reporter, information in the mailing lists and CVE lists are limited to one further kernel problem that was reported by the same reporter, using the same private mail address without company affiliation. Also, the earlier problem (kernel.org, 2022r) this reporter has reported was not assigned a CVE and not explicitly determined as a security problem. The reporter was not involved in the discussions.

The person opening the discussions with the first *patch* draft is David Ahern (kernel.org, 2022o). According to the MAINTAINERS file in the git repository I cloned for the project at the state it was at the time of this CVE's processing, he is one of three maintainers of the related "NETWORKING [IPv4/IPv6]" *sub-system* (and of further *sub-systems*). According to his public LinkedIn profile (available to everyone without logging into LinkedIn), he is himself not employed by the *Linux Foundation* but a third party company that is not in the *Linux Foundation's* platinum or gold members list, but related to Linux engineering.

Involved in the discussions are also Ido Schimmel from NVIDIA (no plainum or gold member), one of two maintainers of the "MELLANOX ETHERNET SWITCH DRIVERS" *sub-system* (this project cannot evaluate if the connection of the mentioned *sub-systems* goes beyond the general connection between Ethernet and IPv4), and Paolo Abeni from Red Hat (Platinum member), one of the four maintainers of both the "NETWORKING DRIVERS" and "NETWORKING [GENERAL]" *sub-systems* (equal four maintainers for both).

4.2.2. The used email service of maintainers: inconspicuous, but still security-relevant?

An interesting note at this point is that both the mailing lists and the MAINTAINER file indicate that there does not exist a formal or informal institution about which email infrastructure to use: some maintainers use a @kernel.org email address, which belongs to the *Linux Kernel Organization, Inc*, whereas some in the previous problem used their employers' email addresses (*Linux Foundation*, *Red Hat*, *NVIDIA*). However, other maintainers of the relevant *sub-systems* use their private email addresses (e.g., with private email domains, such as their own name as domain name).

On one hand, it can be argued that this is not security relevant since *OpenPGP* can ensure end-to-end integrity in the mailing lists. On the other hand, it indicates another point: there is no standardization in the local hardware and software. Indeed, it cannot even be ensured that all these companies deploy hardware/software against some standard. Such a standardization becomes even more unrealistic when people can use their private email domains (which is unlikely to be hosted by their employer). The local infrastructure of developers and maintainers is thus highly heterogeneous.

However, this links back to the institutions of *redundancy* and *trust developers, not infrastructure*: on one hand, there is no one standard/code that can be broken in terms of "find a bug in the standard/code and you have everything that deploys this standard/code". There are many individual infrastructures, and given the previous elaborations, each has only a limited value (because of limited impacts) and even if one is captured, it is unlikely that noteworthy attacks can be conducted: attacks are likely to get revealed at one of the many stages (which are conducted on different infrastructures) any code has to pass until it ends up at a compiled kernel on an operating system, which adds *deterrence* for attackers. Indeed, despite being not really dangerous, the community has identified the four hidden malicious *commits* from the Minnesota case within a few days, despite being suggested by then-unknown fake accounts at unknown times, against a Linux kernel that approaches 1.000 *patches* per day (kernel.org, 2023b). In the end, having captured one infrastructure of one person/organization does neither imply to have another one nor does it imply to know about others' vulnerabilities. And in each case, there is a risk to get caught (beginning with the try to get one infrastructure that is related to kernel development).

Of course this also poses problems to an attacker's capabilities for reconnaissance: the possibility to get information about the infrastructure through inexperienced or unwary employees of lower ranks in order to then exploit that information against, e.g., experienced developers, does not exist to an extent that could affect significant parts of the infrastructure of Linux.

With the two determined security by design characteristics in mind: there is no single point of failure, much balancing possibilities, much "getting caught" possibilities, and any possible unintended input (socially and technically) is unlikely to be applicable to a significant extent on the Linux architecture. Beyond the limited reconnaissance possibilities, attacks are made further unattractive through the *deterrence*. From a market perspective, it can be said that the market

for attacks has a high "entry barrier" given the *deterrence*, whereas any "alternative" is attractive given the low impact and the low chances of success of an attack. In short, the return on investment is low and does not justify the risks of failure and risks of getting caught.

Nevertheless, it has to be said that this security by design result holds for the kernel, where only integrity is relevant but not confidentiality. Yet, even in environments where confidentiality of data has to be ensured, some of the advantages still apply: e.g., limited value if an individual social/technical system is captured, and limited reconnaissance possibilities. Yet, if the focus is narrowly on a very piece of confidential data, any system from which this confidential data can be obtained acts as a single point of failure. However, it has to be noted that informal institutions tailor to changed environments just like formal institutions, and the need for confidentiality in many respects can decrease within societies and organizations: the massive availability and accessibility of information of people with limited possibilities of keeping previously private information secret can be argued to have already led to cultural changes: people today publish much information about themselves, making things public that used to be considered private. This is obviously a probabilistic argument and not yet deeply researched or proven, but the point that has to be derived is that what is confidential at one time and in one context is not necessarily confidential in another time/context, and both formal and informal institutions develop and change over time in conjunction with each other (including social and technical institutions).

Indeed, this could be linked to the "privacy paradox", whose research was summed up and evaluated by Kokolakis (2017): the "privacy paradox" could be the symptom of a transition from traditional "privacy" notions that people have still in their minds when explicitly asked about privacy because they had "learned" in their past environments that this is what they have to consider as "private", but at the same time, the socio-technical developments in societies and systems have already made "privacy" develop, and what people actually treat as "private", is already something different: if traditional privacy notions/interpretations are no longer competitive for people and organizations, they will replace them intuitively, and over time, they will also adjust what they used to have "learned" (and "teach"/"educate") to call "privacy": "privacy" could develop and (if not already occurring) be first informally, and later formally reinterpreted by developing and step-by-step-accepting its adjusted notion(s), comparable to the development of "sovereignty" over the eras. Such developments are not necessarily linked to the age of people but can affect a society and its dynamics as a whole, and thus, indications for age-independent concerns about privacy issues like that of Hoofnagle et al. (2010) do not automatically reject this possibility: architectures like Linux could already indicate where the transition is developing to, whereas some professional fields (like the Linux kernel and the contributions to the actively, intentionally developed "anarchy" around the Internet) that used to be private in multiple respects have in competition already developed to be public. However, any argumentation in this direction is only an implicit possibility and can be neither explicitly indicated nor proven in this project, and deeper analysis of this phenomenon or links/connections/interrelations are out of scope for this project.

However, this leads to the question about how far the Linux architecture and its institutions are examples of what other architectures are also going to tailor to if they just experience sufficient competition (and how far these architectures' environments thus adopt comparable institutions about what is considered confidential). Indeed, before Linux, the widely deployed operating system kernels used to be traditional products where the source code was the major confidential secret: future research might consider the question how "confidentiality" develops (how/when it is constituted, defined and manifested; analysis of the institutions of a society/organization that shape this society's/organization's "confidentiality") in modern environments because this is related to the needs of the security of CIA.

4.2.3. The discussions and its patches

The mailing list discussion documented on lore.kernel.org (kernel.org, 2022o; 2022p; 2022q) is comparable to the previously analyzed discussions. In this case, three *patches* have been discussed (v1, v2, v3) until v3 was accepted. It contained elaboration about which code (of which files) could be affected, the behavior was tested by several

developers and it was considered how to code the *patch* to simplify backports. The discussion contained more details compared to the first problem's discussion, which leads to the question if it reveals more about the socio-technical Linux architecture.

Unlike the first problem, there is a discussion about different possibilities, which leads to questions about a hierarchy (who decides which possibility to implement?). Generally, the hierarchy occurring at this point is not among people but among the *sub-systems*. Obviously, if something is *merged* into *sub-systems* at some point, the related *sub-system* maintainer(s) has/have some authority because they decide what they *pull*. However, the previous elaboration already illustrated that systems or *sub-system* can also accept *patches* from entities multiple stages earlier directly (which means to skip some stages in between). Another hierarchy is neither documented nor indicated: maintainers discuss their respective *sub-system's* needs (kernel.org, 2022o) and have to align with each other according to that (a more detailed elaboration of/assumption about informally accepted/established hierarchies cannot be derived from these discussions). In the end, it can be said that only the maintainer of the *mainline* and the *stable* branches are inevitable authorities, at least within the kernel community itself (excluding the operating system communities).

Nevertheless, in any case, it can be argued that the public code and its legal circumstances can be *forked* at each time, making it formally very easy to create another architecture that could replace any maintainer including *mainline*/*stable*. Yet, how, when and if this is realistic goes beyond the scope of this project. This argument does explicitly not apply to intended forks that remain connected and in interaction with the Linux architecture and that can be considered as part of the Linux architecture (e.g., Android, which is forked regularly from long term branches, is adjusted to its respective needs but also contributes to the actual Linux kernel, making this a reciprocal contribution).

The v2 can be ignored because it was rejected by its author who proceeded then with v3. The v3 was reviewed and tested by another developer and then passed so that it could then be evaluated and tested for the *branches* of the affected kernels. The v3 *patch* was then applied to *mainline*, 5.4, 5.10, 5.15 and 6.0 (kernel.org, 2022s).

Generally, the *patch* was developed by a maintainer of the "NETWORKING [IPv4/IPv6]" *sub-system*, who also adjusted it themselves based upon a please from a maintainer of the "NETWORKING DRIVERS" and "NETWORKING [GENERAL]" *sub-systems*, and after being reviewed and tested by a maintainer from the "MELLANOX ETHERNET SWITCH DRIVERS" *sub-system*, it was the latter who passed it to the mentioned *stable* kernel branches. A more detailed or more formal hierarchy is neither documented nor can it be distinctly derived.

Yet, at this point, this problem differs from the first problem, which was handled from a higher level and involved from the beginning the maintainers of the affected stable branches, although it could not be clearly derived if the creator of the *patch* of the first problem acted as normal developer who suggested a *patch* or if he acted as maintainer of the *mainline* branch. Yet, the latter illustrates again the informality and flexibility in these institutional dynamics.

However, the then-latest maintained 5 era *branch* was 5.19, which was not updated. After this *patch* has been accepted, two further 5.19 kernels have been released: 5.19.16 and 5.19.17. Both do not contain the *patch* (kernel.org, 2022t; 2022u). Yet, it can be questioned if this *patch* can be easily exploited. And at that time, many Linux distributions have been already migrated to the 6 era kernels, which had been released with 6.0 before 5.19.16 and 5.19.17 (kernel.org, 2022t; 2022u; 2022v), whereas 5.19 was not considered for long term support. The exact decision and its reasons are not documented but 5.19 was maintained only to enable a smooth switch to the 6 era, whereas the majority of distributions seem to use either a long term support kernel or have already switched to the 6 era. However, this also illustrates, again, the informal character of decision-making.

4.2.4. The mentioned Linux operating systems

Yet, the last paragraph relates to the Linux operating systems that are involved on the CVE page. Generally, my analyses were equal to the first problem without relevant differences, except one: the Fedora project Linux still maintained releases with 5.19 kernels (Fedora, 2022f; 2022g; 2022h).

However, the maintainer of the kernel within the Fedora community manually added the related *patch* already to the 5.19.15 kernel before passing it to Fedora's testing institutions (Fedora, 2022f; 2022g; 2022h), which solved the issue for Fedora's 5.19-based releases: the update system of Fedora contains two 5.19.15 kernels, the later is with the added *patch* (Fedora, 2022o). This illustrates the informal flexibility among the communities within the Linux architecture, and the architecture's capabilities to tailor and to balance. However, it also illustrates the trust in people given the limited documentation of related decisions, which indicates that trusted individuals can exercise a lot of power in making decisions, creating outcomes others have to tailor to. Yet, the public availability of any data and information (and the legal possibility of forking) limits exploitation possibilities: anyone can be replaced by its community, and if the community or its sponsor does reject that, the community as a whole could be replaced.

With this in mind, it makes sense to involve a case I already elaborated in my first thesis (Klooz, 2021, p.54): in 2020, major decisions by Red Hat about CentOS, another Linux operating system which is sponsored by Red Hat, led several community members to "fork" the Linux operating system CentOS. The community around CentOS back then split into AlmaLinux, Rocky Linux and the existing CentOS. Both remain maintained and independent from the original CentOS project, which also still exists with the changes imposed by Red Hat. Both AlmaLinux and Rocky Linux have been subordinated to formal Foundations in the US and have both different sponsors including large companies like Amazon, whereas AlmaLinux has already partly replaced CentOS at the European Organization for Nuclear Research (CERN). Therefore, it already exists an example about Linux operating system forks when parts of a community have the perception of abuse or inappropriate behavior. Yet, the dynamics of the Linux kernel community are not equal to that of the communities that derive operating systems: it is not possible within this project to elaborate how far this is indicative. Yet, with the assumption that all of them belong to the Linux architecture, it might be additionally noted that this example also illlustrates the relevance of more traditional competition issues: the forking entities had to get sufficient funding. Yet, deeper analysis with regards to fundings, which can create strong incentives and shape institutions, is out of scope for this project.

With regards to the Fedora elaboration, it has to be noted that the two CVEs I elaborated have both been applied to the same kernel within the Fedora Project. Therefore, my involvement in this kernel's regression and general-working testing applies to the second problem, too. However, as elaborated in the related section of the first problem, this does not result in biases or conflicts of interest. When determining the CVEs, I was not aware of that issue.

With regards to other Linux operating systems mentioned on the Mitre page, Debian had back then releases with the *Long Term Supported* 5 era kernel 5.10, the older *Long Term Supported* 4.19, and with 6.1. Yet, it has to be noted that at the beginning of my analysis, one of the deployed kernels had not been fixed at Debian (Debian, 2022c before the introduction of 5.10.162): one update-repository of a Debian release was not fixed before 5.10.162 (Debian, 2022c), a kernel that was released on the 4th of January 2023 and introduced to Debian later, although the fix was already contained in 5.10.158 (kernel.org, 2022w). The kernel 5.10.158 and the kernels from 5.10.158 up to but excluding 5.10.162 have not been introduced to this update-repository. Yet, 5.10.158 was available already in another repository of the same Debian release, where it was already available soon after the release of 5.10.158 (which was before my analysis) (Debian, 2022c).

By default, this Debian release has both mentioned update-repositories enabled so that users would have got 5.10.158 initially and later 5.10.162, each on time (I tested this myself by installing the respective Debian release and verifying its default update configuration). Nevertheless, this illustrates again the flexibility in and the differences between Linux operating system projects: when the Linux architecture is considered as a whole, it has to be clear that the operating system projects need to ensure internally security by design and security in general in some scenarios/ circumstances (such as providing kernels with security-related *patches* on time in some way) because their conduct cannot be balanced by the kernel community in such scenarios/circumstances.

However, in the given cases, the major institutions are widely the same: it are trusted developers with a corresponding *reputation* who conduct publicly, with public documentation of each event and a potentially-high likelihood of massive review and testing (e.g., by experienced users, other developers, and other communities who "fork" from each other).

Much standardization in between Linux operating systems and the public availability of information about their conduct ensure easy ways to identify inappropriate institutions and also low entry barriers to migrate to another Linux operating system, creating incentives to maintain institutions that fit the Linux architecture.

Just like in Fedora and the kernel, much trust lies in some developers and not every conduct goes along with documentation of their reasons, but there seem to be common behaviors in reoccurring contexts, which are taken for granted and thus, do not need dedicated documentation each time they occur. Again, when comparing developers, the boundaries of the communities and projects are fluid/blurred. A more sophisticated analyses of the informal acceptance of when documentation and discussion is necessary and when not goes beyond the scope of this project, but as elaborated earlier, computer and Internet access is sufficient to start a discussion, and then it is the institutional dynamics and the competition of opinions that determine if the discussion develops towards changes or if it gets lost. It can be already said that this applies to Fedora, Debian and the kernel community itself, whereas the previous analysis of Openwall was not sufficiently sophisticated to derive an indication about that.

4.2.5. The "problem following" approach and the two CVE problems' analyses

So far, the two problems that have been analyzed offer valuable indication of the institutions, dynamics and relations in the Linux architecture. However, although the "problem following" approach remains resilient to biases and ensures a comprehensible and reproducible "constant" in the development of the research, the results of this approach have proven to be more superficial and simple than initially assumed. However, it can be questioned if the issue lies in the approach or if it lies in the two CVE problems, which are time-critical but not complex in their solutions, and if the problem is superficial/simple than the results are inevitably the same. And in any case, it can be already valuable indication derived about achieving security in specific contexts and about needs for further research and consideration in future. It remains to be seen what the next two problems reveal: the next two are security-related but not time-critical, being more complex by nature and having introduced new technologies into the kernel.

5. Non-time-critical and security-relevant: introducing security-technologies to the kernel

5.1. Adiantum

5.1.1. Introducing adiantum

Adiantum was added in kernel 5.0 with 21 related *commits* (kernel.org, 2019):

```bash
6db43410179bc40419a9859ebb333f178a49829d
baa6707381285e68cc472efba58e7e736057aacc
8094c3ceb21ad93896fd4d238e8ba41911932eaf
b71acb0e372160167bf6d5500b88b30b52ccef6e
00c9fe37a7f27a306bcaa5737f0787fe139f8aba
c6018e1a00b5c70610cdfb3650cc5622c917ed17
7a507d62258afd514583fadf1482451079fa0e4d
4af78261870a7d36dd222af8dad9688b705e365e
0f961f9f670e7c07690bfde2f533b93c653569cc
012c82388c032cd4a9821e11bae336cf4a014822
b299362ee48db8eab34208302ee9730ff9d6091c
19c11c97c39f5c6280b4d523ea170ef9a8f7ed12
cc7cf991e9eb54cac7733dc7d8f3a8591ba6e1c3
a00fa0c88774bea9a102fc616598d9ee52765451
059c2a4d8e164dccc3078e49e7f286023b019a98
16aae3595a9d41c97d983889b341c455779c2ecf
26609a21a9460145e37d90947ad957b358a05288
1b6fd3d5d18bbc1b1abf3b0cbc4b95a9a63d407b
d97a94309d764ed907d4281da6246f5d935166f8
aa7624093cb7fbf4fea95e612580d8d29a819f67
de61d7ae5d3789dcba3749a418f76613fbee8414
```

The *commits* `baa670*` and `b71acb*` are from Linus Torvalds (*mainline* maintainer) but only *pull patches* from the kernel developers/*sub-system*-maintainers Ted Ts'o (`baa670` of Ts'o's "FSCRYPT: FILE SYSTEM LEVEL ENCRYPTION SUPPORT" *sub-system* → related to the use of *adiantum* at file level encryption within compatible file systems using "FSCRYPT"; unlike block level encryption that is in between the disk and the file systems) and Herbert Xu (`b71acb` of Xu's "CRYPO API" *sub-system* → add related cryptographic algorithms), which are dependencies to achieve all *adiantum* capabilities/features, and all the remaining *commits* are from the *adiantum* developer Eric Biggers (ibid), one of the two *adiantum* inventors (Crowley & Biggers, 2018a; 2018b). "FSCRYPT" is itself nothing new: the related *commit* only adds the possibility to use *fscrypt* in conjunction with *adiantum* and thus, it is not relevant for this project. The "Crypto_API" *sub-system* contains the kernel's cryptographic algorithms/routines that are used among others in *adiantum*.

Adiantum is intended for storage encryption and can be used in conjunction with the "dm-crypt" (which is a kernel "module" of the "DEVICE-MAPPER" *sub-system* for block level encryption, in between the disk and its file systems, implying the file system being itself encrypted; this is the primary *adiantum* use case) and in conjunction with "FSCRYPT" (kernel *sub-system* for file level encryption, encryption within the file system). At the same time, the algorithms/routines are added to and used from the "CRYPO API". *Adiantum* can be seen as a monolithic (in terms of not modular) set of predetermined/fixed algorithms that interact in a "construction" (as the authors call it) that processes a whole sector at once (*adiantum* is designed with 4096 byte sectors in mind, which is efficient for storage encryption and assumed in the following as default). The "construction" was specifically designed for *adiantum* (the authors called *adiantum* a "construction") (Crowley & Biggers, 2018a): in its primary use case, *adiantum* is an alternative to AES-XTS, which is currently the dominant way of disk encryption. However, the "construction" makes primarily use of a stream cipher to process the 4096 byte sectors, and does not contain a traditional "mode of

operation" that processes blocks (ibid). Also, this "construction" makes use of two ciphers: *adiantum* dictates the XChaCha stream cipher with 12 or 20 rounds, and at the same time, it also makes a little use of the AES-256 block cipher (Crowley & Biggers, 2018b, p.40).

However, AES is only invoked once in order to create one block per sector (so, one 128 **bit** AES block per 4096 **byte** sector), whereas the remaining data is processed by XChaCha12/20 (ibid, p.42). Yet, AES and XChaCha12/20 and their inputs/outputs are interrelated with each other and with a hash (NH and Poly1305 for hashing) within an unbalanced Feistel network (ibid., pp.40-42). The author's call it a "HBSH (hash, block cipher, stream cipher, hash) construction" (ibid, p.40). Beyond the fact that (X)ChaCha and AES are the dominant symmetric ciphers on the Internet (and in its standards), and both well researched and well understood (XChaCha incorporates the security guarantees of ChaCha, differing only by XChaCha's extended nonce), *adiantum* seems to generally focus on adopting existing security guarantees with the goal to make it dependent only on the existing security of well understood and proven algorithms and structures. The *commit description* from Eric Biggers in the major *commit* of *adiantum* (`059c2a` → *commit* title "crypto: adiantum - add Adiantum support") contains among others: *"... Adiantum is provably just as secure as HPolyC, in fact slightly **more** secure. Like HPolyC, Adiantum's security is reducible to that of XChaCha12 and AES-256, subject to a security bound. ..."* (kernel.org, 2019). Added to the kernel in 2019 and published also in cryptographic community (ibid), *adiantum* remains considered secure the way it was added.

Obviously, the choice of algorithms and operations (with minimal use of AES) are highly efficient on software implementations while the construction is tailored to the needs of storage encryption. Indeed, *adiantum* solves a security issue of the currently dominant XTS mode of operation: in *adiantum*, an attacker cannot achieve informations about plain text and cipher text relations because in *adiantum*, any one bit change in plain text changes the whole sector (4096 byte), unlike XTS, where one bit changes only affects 16 bytes (Crowley & Biggers, 2018a). Yet, it has to be noted that the revealed information in AES-XTS is on itself unlikely to be exploitable and AES-XTS is from the perspective of security of confidentiality considered sufficiently secure.

However, the major goal of *adiantum* is the unsuitability of existing storage encryption methods for entry/low to mid level hardware (especially mobile hardware) and other systems without cryptographic hardware acceleration. This includes many smartphones and tablets (and smartwatches, TVs, and so on), which often lack hardware acceleration especially for AES. Indeed, previous approaches to storage encryption can be argued to be no increase of security but only shift of security on such devices: the amounts of data that have to be encrypted and decrypted in storage encryption are much higher than, e.g., when opening a web site. Thus, the battery life time of mobile devices can be decreased largely if encryption/decryption is not efficient. However, at the moment, the dominant AES is very slow and inefficient in software implementations, and the XTS mode of operation enforces that any single bit of data has to be processed in its block cipher. Thus, AES-XTS can be seen as decreasing the security of availability while increasing the security of confidentiality, which is a shift of security and no general increase.

However, mobile devices can be stolen and often contain confidential data. With *adiantum*, *Android* (more precisely, Google) started to generally mandate storage encryption by default in devices with Android 10 or later (Google, 2019), suggesting to use *adiantum* without and AES-XTS with hardware acceleration, but mandating one of the two for each device. Before that, sufficiently secure storage encryption had to be AES-256-XTS-based or AES-128-CBC-ESSIV-based and was only mandated by *Android* for devices that had an encryption/decryption throughput above 50 MiB/s (Google, 2019) because of the disadvantages. Without AES hardware acceleration, 50 MiB/s throughput of AES-XTS already requires capabilities often not achieved by average mobile hardware.

When I started a discussion about making *adiantum* the default for Fedora installations on systems without hardware acceleration (this was over three years after *adiantum's* adding to the kernel and related only to Fedora Linux; I have no affiliations or relations to *adiantum* beyond starting this discussion), a developer created a benchmark comparing AES-XTS with *adiantum* on a system without AES hardware acceleration: with 256 bit keys, AES-XTS has only **83.5-84.3 MiB/s**, whereas *adiantum* with XChaCha20 achieves **327.8-345.0 MiB/s** (Fedora, 2022p). The exact hardware is unknown so that the comparison does not offer data about the efficiency and performance in general, but because the

benchmarks of AES-XTS and *adiantum* are equal and on the same hardware, they indicate the performance difference between AES-XTS and *adiantum*. However, this type of comparison (and the way it is documented) should be seen only as a rough indication and should not be considered as proven or representative. Although it is not automatically proven that this performance difference can be transferred 1:1 to the power consumption, it is indicative for the power consumption: if the encryption/decryption throughput can be increased to 393-409 % at the same CPU load, 100% can be likely achieved with correspondingly less CPU load and thus, less power consumption (with regards to power consumption: this can be seen as roughly indicative as long as only the software implementations are compared). It might be noted that the performance difference will be even bigger if XChaCha12 is used.

So, from that point of view, it can be said that *adiantum* brings storage encryption to markets and/or market segments where it could not exist before, including developing markets (Crowley & Biggers, 2018b, p.40). Identifying how and when this indirectly enables further use cases and possibilities with the related hardware in conjunction with storage encryption (with security of confidentiality that does not noteworthy decrease security of availability) goes beyond the scope of this project.

5.1.2. Adiantum, its commits, discussions and the kernel 5.0

The *commits* already indicate that this is much more interrelated and complex than the CVE problems. Indeed, three *sub-systems* are affected and *adiantum*-related code is added to all three of them, whereas the major *commit* "crypto: adiantum - add Adiantum support" (`059c2a`) was added to Herbert Xu's "CRYPTO API" (kernel.org, 2019; Linux git repository), where it became available to be used by "DEVICE-MAPPER's" dm-crypt module and by "FSCRYPT" in conjunction with further algorithms and functions that had been added to the respective *sub-systems*.

As in previous problems, all these *commits* have undergone several instances of review and testing (documented by related *sign-off* tags), in some cases by the respective author and at least one maintainer from the related *sub-system*, in some cases by further developers (kernel.org, 2019; Linux git repository): e.g., the major *commit* (`059c2a`) was also reviewed by another developer (Ard Biesheuvel from Linaro) who is maintainer of several other *sub-system* around drivers and EFI/firmware. The latter was also involved in several other *adiantum commits*. Other people who reviewed/tested *adiantum commits* have been Martin Willi (has only acknowledged some *commits*) from the *strongswan* project (a widespread IPSec implementation) who is himself no *sub-system* maintainer.

The *mainline* maintainer (who added *adiantum* to *mainline* in kernel 5.0) *pulled* the related *commits* from the related *sub-system* maintainers and not directly from the original developers. Indeed, this ensures the *chain of trust* and adds security in conjunction with the contained *patch* that cannot be changed without highlighting the fact in *pulls*: after review and testing in the different stages and *sub-systems*, this ensures a comprehensible and reproducible documentation of the "way" the given code has passed. Many entities are passed that can review and test. Later, once the *patch* ended up in the respective kernel maintainer's *branch*, the original developers (so, the initial stages of the architecture) will *pull* from the maintainer's *branch* to get the new "overall" state of the *branch*, and if their *patch* had been changed in between, this will be highlighted by their *git* when *pulling* so that all related developers will see it.

Within the respective *trees* that are passed by the *patch*, the *SHA1* hash places the respective *commit* that transports the *patch* correspondingly to the next stage, whereas the original *SHA1* hash remains documented in the *commit messages* to make its origin comprehensible and to make its "way" easy to reproduce.

It has to be noted that, correspondingly, different *adiantum*-related *commits* had to pass different entities and pathways before they ended up in a *release candidate* of the kernel in order to be tested in conjunction with each other until the respective kernel (in this case, 5.0) was released.

The discussions around the *adiantum patches* and the *commits* that transported them down to the final kernel are comparably to previous elaborations, but there are multiple *patches* with multiple discussions in parallel. The latter contained several further developers and *sub-system* mailing lists that have been involved, developing *adiantum's* patches up to v4 until acceptance (primarily: kernel.org, 2018). Further, it included Paul Crowley, who is the cryptographic designer of *adiantum*, which is noted in the *commit* description of the major *commit* (kernel.org, 2019).

However, it is also clear that the v4 *patch* is the final that ended up in *adiantum's* major *commit*, whereas the earlier *patches* are the preceding technologies (before v3), not named *adiantum*. The latter are out of scope, although it is interesting and noteworthy that the meaning of the version differs among technologies in the Linux architecture. This adds a limitation because all these preceding discussions of preceding technologies culminated in the *adiantum* technology, which means that these discussions' outcomes are already contained in v4 without dedicated analysis in this project. However, adding all that to the project would change the constants of the *adiantum* analysis and thus, corrupt its comparability to the other analyses. Additionally, it would go beyond the capacities of the project.

Yet, that the "problem following" led to out of scope discussions and that it revealed the inconsistent and flexible overlaps and delimitations of the versioning and the different technologies (such as different technologies that are treated as different versions of the "same" *patch*), and how they link to and result in each other, illustrates the unbiased nature of my approach.

The original message that opened the *adiantum* discussion makes clear that Biggers and Crowley have been working jointly on that to enable *Android* with *adiantum's* capability. Both work for Google, which is the major entity that drives the development of *Android*. However, on one hand, Linux became officially involved not before the discussions. On the other hand, Google had no possibilities to enforce *adiantum's* acceptance: it is a Gold (not Platinum) member of the Linux Foundation, but this does still at the best help to achieve a place at the board, not to impose decisions, and it is limited to the Linux Foundation. Further, the Linux Foundation's power to enforce any will against the kernel community is limited to stop funding those members who are Fellows (who are well reputed and demanded leading developers around the kernel, within an institutional environment where they are responsible for their actions personally, limiting possibilities for successful external influences). In any case, the discussions make clear that the developers are lobbying towards the kernel community by technical arguments, but while they developed *adiantum*, they could not automatically assume that *adiantum* will be *pulled* to the kernel. Nevertheless, Google would have still be able to add *adiantum* to *Android* itself, even if the kernel from which *Android* is forked does not contain it.

The public repository of the two *adiantum* developers is on GitHub (Crowley & Biggers, 2018a), which is a place of publishing but the history of the repository makes clear that it is not the place of development. The development before the time of the first discussions about adding *adiantum* to the kernel is not documented and possibly has taken place internally.

5.1.3. Excursus: what the results of the previous sections in conjunction with my research approach indicate

However, with the last sections in mind, it more and more becomes indicated that my "problem following" research approach applied to technical Linux "problems" creates no immediate relation to non-technical organizational elements. Problem solving, discussions, reasoning and decisions around the kernel can be clearly identified and followed, but the sole "penetration" of the architecture beyond the self-organized kernel community was the "Minnesota" case, where the *Linux technical advisory board* was involved. However, the latter is itself composed only from Linux kernel-related technical personnel and has explicitly not intervened in the decision of the kernel community, taking its self-organization for granted.

However, the absence of impacts and interrelations to non-technical organizational elements when applying my research approach to the kernel's problems can be seen itself as indicative: non-technical organizational elements such as in the Linux Foundation seem to be solely supportive (e.g., fund Fellows in a "neutral home", pay preferred developers to contribute) but not invasive. The public and open processes and decision-making makes outgrowths from individual companies/organizations not realistic.

However, as I already found out previously, there are some decisions in communities around Linux that are not always explicitly explained or justified by the responsible developers. Nevertheless, these are public and work out only as long as these decisions' outcomes correspond to the will of the community, while in this institutional environment, these developers stand with their names for their decisions: the competition in the community strongly regulates itself, although it is not a closed but an open system that is comprised of personnel from many organizations/companies, which each provide incentives directly and indirectly.

Additionally, based upon the CVE problems evaluations, decisions without explanations and reasoning have so far been limited to decisions that are ordinary, occur regularly, being "informally standardized" and generally treated that way, which includes a limited margin for such decisions.

Yet, although the extensive absence of "non-technical organizational element" interrelations and impacts in technical "problem solving" is itself an indicating factor and thus a relevant information, it will possibly pose problems to the implementation of objective 5 of this project, because the comparability of the architectures considered in objective 5 will be limited: if the development of the project does not change, it will be possible to only compare the kernel development (*distributed*) infrastructure with the affected Solarwinds infrastructure (without a wider/holistic organizational comparison and without comparing origins) because every wider comparison would need to fill a lot of gaps of non-technical aspects with biased assumptions. This would also mean that objective 5 possibly will not add much value to research given that there is already some evaluation of this restricted perspective.

Still, the boundaries of the Linux architecture we have seen so far are indicative and the extensive absence of interrelations to the wider non-technical organizational elements at the technical "problem solving" levels provide information about the Linux architecture and about interrelations. It can be said that this strong self-organization, while non-technical organizational elements being supportive only, is itself another institution: a *non-intervening* institution: trust the competition of the community.

If the project keeps developing the way it does, the interrelations of/to the wider/related "eco-system(s)" (as the Linux Foundation calls it) within the socio-technical Linux architecture, and the reciprocal incentives and interactions, have to be analyzed in future projects by defining other types of problems that have to be followed (which can maybe derived from this project's results). However, the information this project has achieved so far about the socio-technical Linux architecture is not less than anticipated, it is only different information about this architecture compared to what was anticipated initially: the focus on people is very strong and their employers, organizations and corporate relationships have only a weak impact and influence (and thus, interactions) in their technical problem solving (again: this is itself indicating information!).

Beyond objective 5, it cannot be said how far the negative impacts on research results also affect the objectives 3 and 4.

5.1.4. Adiantum's interrelations within Linux: uncovering more of the architecture

Adiantum is no *sub-system* but belongs to the "CRYPTO API" *sub-system* and is related to "DEVICE-MAPPER" and "FSCRYPT", whereas "DEVICE-MAPPER" is the foundation of *adiantum's* primary use case: provide block-level encryption.

Differently than with "FSCRYPT" and "CRYPTO API", in "DEVICE-MAPPER", *adiantum* affects only a "module" that belongs to "DEVICE-MAPPER": "DM-CRYPT". As the name suggests, a "module" is a modular component that can be loaded and unloaded to the kernel, based upon demand.

The "DM-CRYPT" module is not developed within the kernel.org infrastructure but on GitLab (Broz, 2020;2022;2023). It is developed along with the utility *cryptsetup* that acts as its interface to setup disk encryption in the realms of "DEVICE-MAPPER" and that sets up "DM-CRYPT" together with its extensions "LUKS" and "LUKS2" ("Linux Unified Key Setup"), which add several capabilities to "DM-CRYPT's" disk encryption such as multiple passwords for one encrypted disk or *Argon2* password hashing (Broz, 2020;2022;2023). "DM-CRYPT's" maintainer is not in the kernel's maintainer list since it is no *sub-system*. "DM-CRYPT", its extensions and *cryptsetup* are maintained by the same person (Broz, 2020;2022;2023).

This leads to the question if this adds attack surface to get malicious code into the kernel, with GitLab being itself an attack vector to the kernel. However, even if developed/maintained on GitLab, any code has to be send to the related Linux mailing lists, discussed, and what was sent to the mailing lists (being already in the kernel.org infrastructures and at the *git* repositories of the related developers) has to be accepted: once accepted, the *pull* is not done again from GitLab. Instead, it is used what has been sent to the mailing list, discussed, and already added to the other developers' and maintainers' *git* repositories. If the code would be manipulated at any instance/entity, this would cause conflicts and highlights on the other *git* instances/repositories in the *chain of trust* as already elaborated earlier. With this in mind, from a security perspective, GitLab (more precisely, the related GitLab repository maintainer) does not differ from other people and entities when suggesting *commits*/*patches* to the kernel.

However, this leads to another interesting part of the Linux architecture that looks at first glance like a blurring factor: *cryptsetup*, finally a tool that can be used on (and that is installed by default on most widespread) Linux operating systems but that is itself not part of the kernel, has its own mailing list on kernel.org (kernel.org, 2023c). Unlike *cryptsetup*, "DM-CRYPT" or "LUKS"/"LUKS2" have not. Obviously, "LUKS"/"LUKS2" is an extension to plain "DM-CRYPT", which might make sense to be not handled separately. However, the "module" that is part of the kernel (more precisely, part of the "DEVICE-MAPPER" *sub-system*), is "DM-CRYPT", not *cryptsetup*.

Yet, as stated above, the development of "DM-CRYPT" (incl. "LUKS"/"LUKS2") and *cryptsetup* is conducted in conjunction with each and commonly on GitLab (Broz, 2020;2022;2023): there is only one repository on which development takes place, incorporating "DM-CRYPT", "LUKS"/"LUKS2" and *cryptsetup*, whereas the repository is named "cryptsetup": therefore, *cryptsetup* is not just the name of the tool that brings "DM-CRYPT" and "LUKS"/"LUKS2" together and manages them in the "DEVICE-MAPPER" *sub-system*. Instead, it is also the name of the "common project". Additionally, it has to be noted that *cryptsetup* also provides libraries ("cryptsetup-libs") that are a dependency to make use of related "DM-CRYPT" capabilities also in some use cases that do not require/allow the use of the utility "cryptsetup" itself.

Therefore, the current mailing list "cryptsetup" is the related kernel.org mailing list for everything around "dm-crypt", "LUKS"/"LUKS2" and *cryptsetup* (kernel.org, 2023d). When reviewing the initialization of this mailing list, it gets obvious that it replaced and succeeded the previous "DM-CRYPT" mailing list, which had a comparable purpose but is no longer active (but still archived) since it was replaced by the "cryptsetup" mailing list (kernel.org, 2022x; 2023d). However, this also illustrates that the mailing lists are not rigid or limited to the very code that is in the kernel, but also consider the outreaches and interrelations in the kernel's deployments/use cases: the constants that determine the boundaries of this mailing list (and likely of other mailing lists and other parts of the Linux architecture) are the needs/impacts of/on the product/use case, which implies the determination is set by the end and not its means. If the product/use case changes, the boundaries do as well.

As elaborated earlier, *adiantum* itself is related to and used in conjunction with "DM-CRYPT"/"DEVICE-MAPPER", but is itself not part of this *sub-system*. However, given its relations and impacts, a review of the "DM-CRYPT" / "cryptsetup" mailing lists reveals that at the time, *adiantum* was also discussed in these mailing lists (kernel.org, 2022x). However, this was just clarifying some questions about the performance in different circumstances and took place after the release of kernel 5.0. Beyond other developers, this short discussion contained the current maintainer of "DM-CRYPT"/"LUKS"/"LUKS2"/"cryptsetup" (hereinafter, "cryptsetup" in the sense of the consolidated project that includes "DM-CRYPT"/"LUKS"/"LUKS2"/"cryptsetup" is abbreviated to "cryptsetup" project).

Yet, the mailing list contains an announce (in kernel.org, 2022x: see "Mon, 3 Dec 2018 10:40:17 +0100") with the release notes of *cryptsetup 2.0.6*: this indicates further testing and reviewing around *adiantum* since *cryptsetup 2.0.6* was explicitly adjusted to "allow *adiantum* cipher construction" in this release. Nevertheless, it cannot be said if only the interface of cryptsetup was developed and tested to make use of *adiantum* or if this included further testing/review of *adiantum* itself, but the "unknown" still adds value through institutions like the *deterrence*.

However, beyond the *sub-system* maintainers, we have further maintainers who administrate and moderate their mailing lists (such as lists of "modules" and related code/tools) themselves, which trigger developments towards the *sub-systems* they belong to: such as the "cryptsetup" project maintainer. The documentation in GitLab around the "cryptsetup" project shows that there have been further "cryptsetup" maintainers before, but they have succeeded each other in taking the responsibility up to the current maintainer: indeed, although it is likely that individual unreliability can be identified early and compensated in this architecture, taking such responsibilities serious is even when it is done informally without legal obligation important and strongly shaping the *reputation* of developers. So far I could not observe a case where critical code was not reliably maintained and if maintainers change, they identify a replacement maintainer before resigning (this can be traced through *git* logs and maintainer files in the different repositories): it is critical that the majority of "critical code" maintainers keeps tailoring to the *hand over before resign* institution, which has been indicated already before. This institution can be also observed in the projects of the Linux operating systems.

5.1.5. Follow Adiantum's development in the 5 era

Adiantum was a new technology, introduced in kernel 5.0. Checking the remaining kernels of the 5 era will reveal how *adiantum* has evolved within the kernel in this era. This relates to questions about its maturity and the quality at the time of introduction (have bugs or problems appeared after its introduction?), its impacts and adjustments (adjustments within *adiantum* or other adjustments with regards to *adiantum*).

Within the previously prepared kernel git repository, I did ...

```bash
git checkout v5.19.17
git log --grep=diantum
```

The first command switches to the last kernel from the 5 era. The second command checks the history (including previous kernels) if *diantum* (to have both "adiantum" and "Adiantum" included) is mentioned within any preceding *commit* in the *commit's* header/topic or in the *commit description*. Since *adiantum* was added in v5.0, a filter for kernels before 5 is not necessary (although I compared against `git log --grep=diantum --since="<kernel release date>"`), and I have reviewed the output to redundantly ensure all belongs to the 5 era: this was the case. All output *commits* have been added in between 2018 and 2020.

The output *commits* have been:

```bash
f000223c981a7c75f6f3ab7288f0be7b571c3644
1c08a104360f3e18f4ee6346c21cc3923efb952e
ba44840747bdd60095881830af0d75f0e5017c99
73bed26f73a120f14cabf8d214ec5078bb42dea9
cd900f0cacd7601dabdd028e8cbdbf2a7041cee2
b9f76dddb1f9f70e008b982381bbc9a67c9b8c66
85af90e57ce9697d36d479124e0bfffb145e39a4
48ea8c6ebc96bc0990e12ee1c43d0832c23576bb
3a2f58f3ba4f6f44e33d1a48240d5eadb882cb59
c1144c9b8ad94d8c11809d75c1f322a853cdfc4a
a4d14e915bcb86e13b45231cd4fe2ce19bd9ba86
adbd9b4dee70c36eaa30ce93ffcd968533044efc
63bdf4284c38a48af21745ceb148a087b190cd21
333e664772c5b8cfd5c019e300b378cd9d1c5ae3
367ecc07314a650084f8a46fe3ce550a9cdabef3
```

- `333e66` and `367ecc` only add test vectors that are related to *adiantum*, whereas the related *pull* from the *mainline* maintainer is `63bdf4`. These three *commits* do not contain changes to *adiantum*.
- `adbd9b` only mentions *adiantum* as an example of how a problem can be solved and thus, is not related.
- `a4d14e` is to improve the value of the output of a warning message by adding the information which algorithm has caused the warning message: this does not belong to *adiantum* but it can affect *adiantum* and also other cryptographic constructions.
- `c1144c` adds a function to "FSCRYPT", which is not *adiantum*-specific but that can be useful in conjunction with *adiantum* but also in conjunction with other cryptographic constructions.
- `3a2f58` adds a function to "CRYPTO API", which improves efficient use of low performance cores (not *adiantum*-specific), which can be useful in conjunction with *adiantum* but also in conjunction with other cryptographic constructions.
- `48ea8c` is only a re-organization of routines that moves some routines into a separated library: this affects some *adiantum*-routines but also other routines. This has no effect on or relation to *adiantum* itself.
- `85af90` fixes an issue that affects encryption modes except *adiantum*: if other encryption modes are improperly deployed, in some circumstances it can happen that the related error is reported not immediately but after the setup. This fix ensures that the error is reported immediately.
- `b9f76d`, `cd900f` and `73bed2` only mention *adiantum* as an example and have no technical relation to it.
- `ba4484` improves *adiantum* by making it use some new functions that had been added to the kernel in the meantime and also simplifies error handling.
- `1c08a1` generally improves the use of 64x64 arithmetic, which is itself not related to *adiantum* but this *commit* had to be adjusted to fit *adiantum*. This again illustrates the previously identified determination of the boundaries of what is considered, which are determined by the end and not the means: the deployable kernel/product/use case.
- `f00022` just mentions *adiantum* within an example of some constructions that should not be used in conjunction with each other: this is technically not related to *adiantum*.

Generally, although being related only indirectly, improving/adding test vectors can be argued to be an *adiantum* improvement but not as security-critical. The *commits* show that in the kernel, *adiantum* has developed within its environment: when new functions have been added that are complementary, *adiantum* was *patched* to exploit the new possibilities. The *adiantum* code developer keeps maintaining his code, and other developers consider *adiantum* when

their own code could affect it in considered use cases: changes have to consider their environment and their interrelations/impacts, and because *adiantum* has been part of the kernel throughout the considered era, other *patches* were adjusted to fit *adiantum*. *Adiantum* develops in its environment, and its environment develops with *adiantum*.

The *pull* from the *mainline* maintainer (`63bdf4`), the non-adiantum-specific arithmetic improvement (`1c08a1`), the non-adiantum-specific routine-re-organization (`48ea8c`) and the non-adiantum-specific "CRYPTO API" improvement for low performance cores (`3a2f58`) are not from the original code/*patch* contributor of *adiantum* (Eric Biggers). Except these examples, all of the above is from Eric Biggers who keeps *adiantum* maintained, comparable to the maintainer of *cryptsetup*/"DM-CRYPT". This finally explicitly indicates another institution, that has been implicitly indicated before, that is well known and that is often referred to on the Internet when people ask about how to contribute to the kernel: if you add code to the kernel, it is assumed that you are responsible to keep it maintained. It remains an informal institution. This *contribution implies maintenance* institution is interrelated with the *hand over before resign* institution.

Even if this is only an implicit interpretation, it could be argued that people are ont just expected to maintain their code, but also that is it expected to not intervene in other people's code without their consent. However, the latter is not explicitly indicated but only a possibility as far as the data of this project suggests it so far.

Generally, this institutional environment provides obviously sufficiently rewards to developers to not just contribute but also to maintain: kernel contributions are highly reputated and well improve a developer's CV: the kernel is itself an important institution of the Internet and of modern societies. Beyond developing code that becomes selected for the kernel, this can lead to "document and certify the reliability" of a developer. Also, the discussions are likely to provide experienced suggestions to contributing developers to improve their capabilities.

At the same time, the entry barrier to suggest a *patch* is low and even if an individual would fail to take the responsibility, the community in its institutional environment is likely to identify and compensate this as long as failing remains an exception. Just like the possibility to enter and open discussions (and tickets), code contributions in this environment are shaped by an implicit *better "enable unnecessary/unmaintained contributions" than "disable relevant contributions"* institution that is the outcome of the agency of the community.

However, given the impact on the *reputation*, and the public and open availability of everything to everyone, intentionally disadvantageous/flippant contributions or unreliable maintenance are likely to be directly or indirectly punished (practically ending up in the "public CV") and thus, discouraged. So the *deterrence* does not only affect attackers. Nevertheless, the informally established institution that avoids *detterence* against contributions at all (e.g., for becoming known for having made an unintentional mistake in a code) is the likewise implicit *mistakes create experience and thus knowledge* institution, which is already known from other realms and not limited to IT (and maybe even a constituting institution of "science" in general). Indeed, in all previous problems, it was the goal to solve them as soon, efficient and reliable as possible: the question whom to blame for an issue was not contained in the data this project has analyzed so far, whereas needs for corrections in the mailing list discussions have been treated just by ideas how to improve: acceptance of people making accidentally mistakes is indicated to be high and seems to be not punished, although this project has not sufficient data to deeply evaluate this institution and how far and where it applies and where not.

"Fault tolerance" and acceptance of making mistakes (and assuming that mistakes are likely to occur when development takes place) have already proven to be institutions that increase security, and enable an organization to not just quickly and effectively solve mistakes, but also to learn and tailor to avoid the occurred type of mistakes in future: even if not always treated as institutions, this is already researched and taught in information security (e.g., the "security management" module of the Royal Holloway's M.Sc. Information Security). So far, the indication leads to the conclusion that the Linux community indeed intuitively differentiates in its institutional environment between unintentional mistakes and avoidable laziness or unreliability. Indeed, it could be argued that the recent paragraphs indicate the scientific origin of Linux in computer science.

5.2. Wireguard

2019-2020 *commits* in kernel 5.6 that reference *wireguard* are the following (kernel.org, 2020a):

```bash
979e52ca0469fb38646bc51d26a0263a740c9f03
c8cfcb78c65877313cda7bcbace624d3dbd1f3b3
3c025b6317272ee8493ee20fa5035c087626af48
11a7686aa99c7fe4b3f80f6dcccd54129817984d
2b8765c52db24c0fbcc81bac9b5e8390f2c7d3c8
a5588604af448664e796daf3c1d5a4523c60667b
551599edbfff2431cef943a772fbde1c3e26eaf8
166391159c5deb84795d2ff46e95f276177fa5fb
3dc55dba67231fc22352483f5ca737df96cdc1e6
82d81bb070cf109b3b5167597239dfa5c45558a5
1fbc33b0a7feb6ca72bf7dc8a05d81485ee8ee2e
175f1ca9a9ed8689d2028da1a7c624bb4fb4ff7e
2a8a4df36462aa85b0db87b7c5ea145ba67e34a8
04ddf1208f03e1dbc39a4619c40eba640051b950
2019fc96af228b412bdb2e8e0ad4b1fc12046a51
a12d7f3cbdc72c7625881c8dc2660fc2c979fdf2
291abfea4746897b821830e0189dc225abd401eb
7bb77d4b8567b35aadd57f3154a08d873572ae20
88f404a9b1d75388225b1c67b6dd327cb2182777
4a2ef721e60f0d2babeb29dbcfe05904b53ea19e
f9398acba6a4ae9cb98bfe4d56414d376eff8d57
ec31c2676a10e064878927b243fada8c2fb0c03c
9981159fc3b677b357f84e069a11de5a5ec8a2a8
bd2463ac7d7ec51d432f23bf0e893fb371a908cd
a8bdf2c42ee4d1ee42af1f3601f85de94e70a421
dcfea72e79b0aa7a057c8f6024169d86a1bbc84b
704a0afb4963a486f604fba1064f9025e2865fc9
736775d06bac60d7a353e405398b48b2bd8b1e54
04d2ea92a18417619182cbb79063f154892b0150
9a69a4c8802adf642bc4a13d471b5a86b44ed434
6f6dded1385cfb564de85f86126f1c054c8f47b1
d89ee7d5c73af15c1c6f12b016cdf469742b5726
43967b6ff91e53bcce5ae08c16a0588a475b53a1
a2ec8b5706944d228181c8b91d815f41d6dd8e7b
d7c68a38bb4f9b7c1a2e4a772872c752ee5c44a6
65d88d04114bca7d85faebd5fed61069cb2b632c
e7096c131e5161fa3b8e52a650d7719d2857adfd
```

When reviewing the changelog, it already becomes obvious that *wireguard* contains much more complex changes to the kernel than *adiantum*. Indeed, *adiantum* primarily added a cryptographic construction whereas the majority of functions have been nothing new ("DEVICE-MAPPER"/dm-crypt, "FSCRYPT", and so on) but existed before, making *adiantum* just a new construction that can be used by and in conjunction with these existing functions.

However, *wireguard* on the other hand, has become its own *sub-system* in the MAINTAINERS file, which remained maintained by the original author Jason Donenfeld (based upon the *git* repository's MAINTAINER file as of 5.19.17).

Indeed, unlike *adiantum*, more complex and subsequent *commits* had been necessary before a *release candidate* for 5.6 could become the official release 5.6: the kernel's *release candidates* had been tested and adjusted by new *commits* each time towards the next *release candidate*, with the last *wireguard* change being made to *release candidate* 7.

5.2.1. Wireguard: from the beginning

The first paper was published in 2017 (Donenfeld, 2017), and has evolved into an implementation that could be added to the kernel and for that early a dedicated mailing list was deployed (wireguard@lists.zx2c4.com, archived in https://lists.zx2c4.com/pipermail/wireguard/). Additionally, before the kernel, further (especially cryptographic) reviews have

been conducted and published, and integrated into the continuing development and adjustment of *wireguard* (e.g., Donenfeld & Milner (2017)). Indeed, the mailing list and development dates back to 2015 as it can be seen in the zx2c4 archive, where the initial prototype had been published.

Given the author's evolving *reputation*, the easy and fast distribution of information in the communities, and *wireguard's* eligibility for many use cases, it has early received widespread attention and thus, contributions, which makes it reasonable to argue that it had already reached a high level of maturity before it was considered to be added to kernel 5.6 in 2020.

5.2.2. Developing commits

As noted above, with its *merge*, *wireguard* became its own *sub-system*, maintained by its original author. But before that, *wireguard* had to pass the kernel community's testing and received adjustments to fit stably into the kernel, but *wireguard* also caused itself adjustments to other parts of the kernel. Indeed, the heterogeneous adding of code from multiple different entities leads to vast possibilities for intended and unintended interactions and impacts of code that can occur in testing, whereas the unintended interactions/impacts need evaluations and adjustments before the kernel can be released.

As in *adiantum*, the *commits* with reference to the technology also contained minor fixes to other technologies that became revealed through the testing in conjunction with *wireguard*. Thus, the persistent developments and additions of code in the kernel with each release also adds new testing data to existing technologies and thus, contributes to the maturity of existing technologies in the kernel and to their evolvement towards critical "enterprise" infrastructure stability/security. Therefore, keeping one *mainline* to evolve over time without re-starting from scratch also adds noteworthy stability/security guarantees. Generally, keeping working on one *mainline* has indeed an impact that shapes the Linux architecture so that it can be determined an institution: *persistent evolvement of one mainline*.

With a focus on *commits* that are part of *wireguard* and its integration, all *commits* had been passed initially through Jason Donnenfeld, who collected and prepared them. Then, everything was merged at the stage of David S. Miller (Red Hat), himself a maintainer of several *sub-systems* (including "NETWORKING [GENERAL]", "NETWORKING DRIVERS", "NETWORKING [IPSEC]", "NETWORKING [IPv4/IPv6]" and "CRYPTO API": all related to *wireguard*). Finally, everything was *pulled* by the *mainline* maintainer to end up in the next *release candidate* and finally, in the stable 5.6 kernel.

Interesting at this point is that all these *sub-systems* (except "CRYPTO API") share a common mailing list, which is the center of related discussions: netdev@vger.kernel.org (https://lore.kernel.org/netdev/). However, *wireguard* was discussed, optimized and developed in several mailing lists. Besides those already mentioned, the "CRYPTO API's" linux-crypto@vger.kernel.org (https://lore.kernel.org/linux-crypto/) is another one. Additionally, there is another mailing list that is more general (thus, overlapping with the other lists): linux-kernel@vger.kernel.org.

The discussions do not much differ to *adiantum* in terms of their institutions, except that they are more complex and interrelated, creating much more dynamics: many mailing lists with different perspectives in parallel to discuss, contribute and test, whereas different subsequent *release candidates* reveal the outcomes, leading to further elaborations in the mailing lists, and so on. The previous paragraphs are related to the previously elaborated determination of boundaries in and among mailing lists and beyond, determined by the end and not the means: the deployable kernel/product/use case.

The people involved in the *wireguard* integration come from multiple organizations from different parts of the world, having no general common constants except their role in the kernel community: a freelancing security researcher (the author), developers from mid-sized companies (two, but without company affiliations in their contributions), Google, Huawei (two), Red Hat, Linaro, Linux Foundation, Linux Kernel Organization, Australian Public Access Network Association Inc. (APANA), and a software security consultancy ("radicallyopensecurity.com"), two further developers whose company affiliations (if any) cannot be explicitly derived (Matt Dunwoodie, Josh Soref), and a PhD researcher from Aarhus University (Mathias Hall-Andersen). Some contributions came from other *sub-system* maintainers

(Krzysztof Kozlowski; Florian Fainelli) and kernel contributors (Eric Dumazet). Additionally, the other "CRYPTO API" maintainer next to David S. Miller, which is Herbert Xu (already known from *adiantum*, from APANA), was involved in some contributions and in the final integration. This also involved the *stable* and *mainline* maintainers from the Linux Foundation. Involvements in testing, reviewing an signing off also included in some *commits*: Eric Biggers (Google; see *adiantum* analysis), Ard Biesheuvel (another *sub-system* maintainer).

Many contributions have been collected throughout the *release candidates* from different people: some have suggested in the mailing lists, some have contributed code, some have reported issues they found: developers, testers, researchers, auditors, and so on. Additionally, it was mentioned that one further private security auditing firm was involved to do an explicit dedicated audit on the *wireguard*-related code before the merge. However, how many have done such audits cannot be identified: we only know one was explicitly hired, and some have identified issues that have been solved throughout the *release candidates*.

Nevertheless, this marks another outcome of the institutional framework: it cannot be foreseen how many have reviewed and tested, and how/what they have reviewed and tested: the above only includes those who actively contributed, which implies they were the first to have found something. This strengthens the *deterrence* institution and obviously, gives a lot of space to improve stability: as already known, there are strong incentives to try to contribute, which starts with taking the *commits* and/or *release candidates* and test and review them.

Additionally, the absence of knowledge of who tested, reviewed, analyzed or audited code (or the technology that is implemented by the code) facilitates a low entry barrier to contribution: testers/reviewers/auditors can decide themselves if they publish their results or not, which avoids any type of humiliation for not finding something. This encourages contribution, and it has to be clear that any audit/review/test is a contribution even if it cannot identify bugs/flaws and remains unpublished. Although being more a passive institution rather than an active one, it can be still argued that this marks another institution, *humiliation avoidance*, that is indeed not existent everywhere: in many realms, getting code on itself ends up in this being formally documented and thus leads to expectations to conduct a test/review/audit, which itself can create pressures and thus deter from contribution.

This becomes further improved by the widespread deployment of public testing tools even before the testing within the Linux operating systems: several bugs have been identified by a *fuzzing* bot from Google, within which the code was tested and that is embedded into testings, being mentioned with the reported-by tag: no one knows how many and which testing tools have been deployed, and with which configurations. Even beyond attacks, this enables many different perspectives and widespread knowledge that are likely to identify bugs of any kind. All results are available to anyone, which again leads to different perspectives and widespread knowledge on the results and corresponding derivations. The availability of massive automated testing means is also an institution, although one that is already known from other realms. Yet, the difference is that such social and technical institutions have not yet been analyzed in conjunction with each other.

Further, the importance of *reputation* and the explicit/distinct embedment of the *reputation* institution can be observed in the changelog of 5.6: it is explicitly and penetratively mentioned if something was reported by someone and by whom. Even if someone just suggested something that was implemented, this was mentioned and documented the same way, by tagging and an additional sentence that makes clear who contributed with what: this is not just about writing the code, but also just, e.g., having had the idea.

In any case, this marks only the final stage of *wireguard* to end up in the kernel from the initial collection of *commits* from the first 5.6 *release candidate* until *wireguard's* last adjustment in the *release candidate* 7 that finalized *wireguard* towards its own kernel *sub-system* and towards the first kernel that officially integrated *wireguard*. This project does not evaluate the "way" *wireguard* has passed from 2015 until this final stage in 2020: getting through all mailing lists from back then is possible and the related information is available, but it goes beyond the scope and the capabilities of this project.

5.2.3. Development over the remaining 5 era kernels

Within the previously prepared kernel git repository, I did:

```bash
git checkout v5.19.17
git log --grep=ireguard
```

"ireguard" is to ensure to contain "wireguard" and "Wireguard". I compared the output against `git log --grep=ireguard --since="<kernel release date>"`. I also checked the most recent output *commit*: it belongs to the 5 era.

However, the complexity and the amount of code related to *wireguard* results in 109 *commits* in the 5 era. This goes beyond the capacity of this project. However, selections of *commits* would introduce biases that would undermine and corrupt the results and the comparability to other analyses/results of the project: being unbiased is a major goal of this project. Therefore, I limit this part to a superficial but unbiased analyses, considering all 109 *commits* but only their meta data.

Generally, *wireguard* remained maintained in the 5 era with regular *commits* with the most recent *commit* being from "Sep 16 15:37:40 2022", which was added to 5.19.12 (5.19.17 was the last of 5 era, released about 1 month after 5.19.12). The maintainer and major contributor remains Jason Donenfeld, whereas the following *commits* have a major difference: because *wireguard* became its own *sub-system* after the release of 5.6, David S. Miller is no longer involved in between Donenfeld and the *mainline* maintainer. However, there are still a few *commits* from David S. Miller (his *sub-systems* are still interrelated with *wireguard*), and also *commits* from some other contributors from Red Hat, Linux Foundation, Meta, Huawei and several further contributors that cannot be explicitly linked to an organization.

However, interesting throughout the *wireguard* analyses is the informal and unregulated use of email infrastructures: some people use their private email addresses/infrastructures, some the email addresses/infrastructures from their employers, some use those from kernel.org or linux-foundation.org, and some switch between different addresses/infrastructures in different *commits*. Indeed, some maintainers of multiple *sub-systems* are referred to by one email address at one *sub-system*, and referred to by a different email address at a different *sub-system*. In this project, it cannot be identified if, e.g., a maintainer is paid to maintain one *sub-system* by his employer but maintains another one privately (while affiliations with, e.g., the Linux Kernel Organization are probabilistic in terms of being private of occupational). However, this correlates to previous indications. Yet, this makes clear that there is often also an affiliation with the Linux Kernel Organization or the Linux Foundation, and given the incentives of reputated developers, it is not likely that the provider of, for example, the next salary has an increased possibility for shaping the development/contribution of a developer/maintainer in a manner that could be disadvantageous for the community. Indeed, the *reputation* and its indirect rewards (which is related to potential salaries at employers in general) are likely to be stronger incentives.

At the stage of the operating systems, after the kernel's release, *wireguard* does not differ to previous elaborations. Yet, once deployed to operating systems, *wireguard* has soon created an impact beyond the kernel in and for multiple use cases, and many user space implementations and additional kernel implementations for other operating systems have evolved soon after, including many products that are based upon *wireguard*. Yet, the wider impacts of *wireguard* remain out of scope for this project, although research about *wireguard's* widespread impacts should consider in what institutional environment *wireguard* has evolved.

6. Non-time-critical and non-security-relevant: are non-security technologies added securely?

6.1. Freesync

As the previous analyses have already revealed, there are no dedicated security considerations *adiantum* and *wireguard*: technologies and code are developed and tested in order to be well understood, as simple and efficient as possible, free of bugs, free of unnecessary functions, and evaluated from multiple perspectives and always in conjunction with everything related. If achieved, this obviously integrates the mitigation of bugs and flaws that can be exploited (security of integrity and confidentiality) and the maximization of stability and expectable behavior (security of integrity, availability and sometimes indirectly also confidentiality).

Dedication of security (implying separated treatment) is conducted only when it comes to bugs/flaws that have been revealed in existing kernels that have been already deployed in production environments and where it cannot be excluded that the bug/flaw could be exploited by an attacker. In this project, this relates only to the CVE problems. Additionally, the separated treatment is conducted only the initial stages of development, which ensures that at the time of publication of the respective bug/flaw, Linux operating system developers have already possibilities to mitigate the exploitation in their respective environments (e.g., besides information and elaboration of the problem, and involvement of reputated people from operating systems, in both CVE cases there had been already an initial *patch* that could be used to help to understand, estimate and categorize the issue, and that might be also used in provisional mitigations until the release of an official kernel *patch*).

With the absence of security dedication in mind, the *Freesync* issue is unlikely to add value but if it does, it would be revealed in the initial stages (where security dedication took place at the CVE problems): a review of *Freesync's* beginning stages shall give indication. As noted earlier, *Freesync* was added with kernel 5.0, just like *wireguard* (kernel.org, 2019).

Obviously, the majority of *commits* are from several AMD employees: *Freesync* was proposed and driven by AMD. The technology's *commit* development is comparable to *wireguard*: there are much more impacts and considerations than in *adiantum* because a whole, new technology was added, whereas *adiantum* is in comparison just an additional construction that can be deployed with existing functions and *sub-systems*.

However, the *Freesync's* development took place on freedesktop.org (ibid), driven by AMD employees. The reason is that the *sub-system*, to which *Freesync* was to be added, is hosted on freedesktop.org (the related mailing list is dri-devel@lists.freedesktop.org, which is related to multiple *sub-systems*). freedesktop.org is to host "free and open source software": beyond kernel *sub-systems*, it hosts much software that is deployed on Linux operating systems, most importantly *systemd* (which is the "init process" on most Linux operating systems, being process "1" that manages the system and its services, and strongly shapes the agency of the operating system including its security/stability). Thus, it is interrelated with the kernel and belongs to the distributed Linux architecture, adding another technical infrastructure that hosts repositories.

However, although it is interesting, this is nothing new and already indicated before (e.g., GitLab). Yet, unlike the commercial and privately owned for-profit GitLab, freedesktop.org is as neutral, transparent and non-profit like the kernel.org's Linux Kernel Organization or the Linux Foundation: indeed, freedesktop.org belongs to the SPI ("*Software in the Public Interest*"), which has been mentioned earlier as an organization that has risen out of *Debian*.

The changelog around *Freesync* does not differ much to previous analyses: most *commits* had to pass a maintainer from another company (Red Hat) to get *Freesync* into the kernel, whereas both AMD employees but also other contributors already adjusted existing technologies in the kernel to make use of *Freesync* (e.g., a contributor from Canonical). Some of the minor adjustments that add *Freesync*-support to existing technologies had been merged directly.

However, although no additional institutions are indicated, it is interesting that the development of the *Freesync* technology itself was done completely by AMD employees (itself not special since AMD announced and drove the technology), but the employees stand and act for themselves and for their respective work each individually, both indicated in the kernel.org logs but also in the *commits* (and related argumentation/discussions) they *commited* on the freedesktop.org infrastructure, respectively. This increases the indication of *adiantum* because the discussions and contributions at the kernel community in the *adiantum* case had been done by only one of the two *adiantum* developers from Google, whereas the other one was not explicitly involved when *adiantum* was to be added to the kernel at the kernel 5.0 discussions. Now, *Freesync* shows the dynamics when multiple developers of a technology are involved in the discussions to get a technology into the kernel.

At this point, it can be said that *Freesync* develops as expected and its analyses is indicated to develop comparable to *adiantum*/*wireguard*: in the subsequent stages, there has been no security dedication even at the security-critical problems. Thus, further analysis of *Freesync* cannot add value to this project.

As expected, a dedicated analysis of the time-critical non-security problem *"mmc: core: Terminate infinite loop in SD-UHS voltage switch"* is dispensable and will be thus not implemented.

7. Conflict actors & the institutional dynamics

Like the two CVE problem analyses, *adiantum* and *wireguard* have added some but little indications and complexities for "conflict actor"-related derivations. Nevertheless, it remains indicative that the covered architecture in the "problem following" is, with an exception when handling the Minnesota case, limited to technically-focused parts of the architecture without much outreach (mostly focused only on the very problem and its very solution). Being itself important information that indicates how the Linux architecture works and creates a stable, secure and competitive kernel and operating systems, it still imposes limitations to the subsequent objectives and limits the indications that can be produced by the analyses that follow the actual "problem following" of the previous parts of this project.

The limitation was not much reduced by *adiantum* and *wireguard* and additionally, except *wireguard* and *freesync*, there is a limited possibility to identify the initial "conflict actors" that have facilitated/provided the initially submitted *patches*/*technologies*, which could then evolve in the public discussions towards the finally accepted *patches*/*technologies*. *Wireguard* can be a bit more indicative because it has been handled, discussed and developed publicly from the very beginning. However, given that *wireguard* is the most complex technology in focus of this project, it has a history of four years before it entered the discussions about a merge into the kernel (this was not expected to this extent in the beginning of the project). *Freesync's* history is more manageable but still sufficient to go beyond the capabilities of this project, especially because of the limited possible value that could be returned for the time investment: the *Freesync* public documentation eligible for conflict actor derivation is less expressive than that of *wireguard* because it has evolved as an AMD-specific project.

On one hand, the number of "problems" analyzed in this project that have proven to be eligible for "conflict actor" analysis *from their very beginnings* (which means the time before they enter discussions for kernel integrations) is limited to two problems (*wireguard*, *Freesync*), and these two problems are not equal based upon the original definitions of the project (one is security-relevant, one is non-security-relevant): this limits the value of the data and of comparability possibilities. Therefore, it is not possible in this project to indicate or even identify what institutional "conflict actor" dynamics do generally apply to the architecture, which are security-specific or non-security-specific,

and which are exceptions that are not representative for the architecture. This limits the results of *wireguard* and *Freesync* "conflict actor" analyses *in their very beginnings* in the conclusions of the project. Furthermore, they go beyond the capacities of the project. Therefore, *wireguard* and *Freesync* will be analyzed only to the extent as the other problems: the point once the discussions about an explicit *merge* into the kernel started.

The concepts and differences among "conflict actors" and "formal actors" have been introduced in the early section "Analysis & stages", primarily between the third and the sixth paragraphs.

7.1. Framing the institutional dynamics

As already elaborated, in today's socio-technical systems and societies, the transitions between social and technical realms are blurred, fluent and omnipresent: boundaries, interrelations and impacts can be neither delimited nor limited to technical or social realms.

The Linux kernel itself is one product of the Linux architecture. However, the kernel itself also is at the core of much of the Linux architecture's technical infrastructure, including the infrastructure of many of the developers and service providers. Nevertheless, in this project, the scope has been limited to the evolvement of the kernel (with the kernel being the product of the architecture). The project did not consider the technical security of a deployed kernel as dedicated object of research: on one hand, the technical security capabilities are widely the output of the architecture and thus, explained by the architecture as far as it concerns the holistic socio-technical perspective. On the other hand, the purely and deeply technical security of existing/released kernels when deployed in production environments has been already undergone more sophisticated research elsewhere and is not the focus of this project: but it has to be clear that based upon the definitions of this project, a *patch* is a technical institution. The latter was not relevant for the goals of the previous objectives, but technical institutions like *patches* or the properties/functions of *patches* become relevant in the "conflict actor" derivation because they become a determinant that delimits "conflict actors" from each other (and distinguishes them).

Generally, both the social and the technical institutions in the Linux kernel (and if, how, when and where to implement/deploy these institutions/functions) are essential in deriving "conflict actors": the Linux architecture institutions as determined in this project are the common constants that enable the "conflict actors" to evolve, whereas (mostly but not solely technical) institutions are determining the "conflict actors" in the discussions: e.g., a "conflict actor" might be defined by the goal to remove functions of a *patch* that are not necessary or that can be *merged* into each other to simplify or to adjust something, to avoid complications in *backports* and so on.

Dedicated evaluation of the technical administration of the infrastructure goes beyond the capabilities of this project. As far as it concerns the project, the related resilience is already indicated to be linked to the distribution of critical infrastructures (instead of centralization): as already elaborated, the Linux architecture assumes any one specific infrastructure (and configuration) within the overall architecture as something that is expected to be breakable anyway.

Nevertheless, it has to be noted that the administration of the Linux operating systems in the server infrastructures (especially kernel.org) could be also indicative given the resilience and importance of these systems: however, research of that has been done by the Linux community itself explicitly and led already to the Linux certification programs "Linux Foundation Certified Engineer" and "Linux Foundation Certified System Administrator", which did integrate security into administration rather than dedicating it as separated topic: this concept has been already part of my previous non-academic publication that is linked to this point (Klooz, 2020), but out of scope in this project. Also, despite being the outcome of the experiences of the Linux community, so far this project's development has not led to explicit indications if these concepts/programs/certifications are also deployed within the Linux architecture. The "problem solving" did neither lead to these administrative parts of the architecture, nor did the project consider the related administrative "problems" as dedicated problems. However, this could develop itself to future research projects.

Generally, it has been already clear in the previous analyses that this is a "collaborative competition" towards a common goal of all participating formal entities, whereas the Linux kernel as a whole is competing against alternatives and against yet-unexploited-possibilities. The collaborative aspect is likely to be rooted in the absence of limited merge slots: developers have not to compete for a limited number of merge slots. Instead, they have only to compete for adding value to the kernel, which then adds value to any related entity. At the moment, it seems that the technical requirements are sufficient to keep the number of *merge requests* limited to an amount that can be tackled. Alternatively, the Linux architecture is capable of tailoring its *sub-systems* intuitively in a manner that balances its capabilities to process *merge requests* accordingly to the number/places of *merge requests*: this project will not be able to indicate which of the two applies. However, the capability to balance requests is likely to be given as long as the number of experienced developers with trusted reputation has a suitable ratio to requests.

The need to break or undermine this competition is low, since there are no entry barriers with everyone being able to participate: any company that needs a derived product can participate to the mature kernel with the requirement of its additions to be valuable. However, the latter is likely if the product has a competitive advantage. Otherwise, the company is not likely to succeed anyway, or it is likely a niche product that is not competing with the kernel. However, at the leadership levels, I already uncovered criticism and issues around *SHA1* and the Minnesota case. It cannot be verified in this project how far there are possibilities for valuable additions that are excluded/rejected because of biases: so far, there is no indication for the kernel community to not follow the demand, and it seems to already determine and provide what will prove competitive tomorrow, whereas alternatives like *Android* keep a "fork" relationship to (and thus, a dependency from) Linux with reciprocal collaboration and code contributions.

7.2. Deriving the conflict actors from the institutional dynamics

7.2.1. CVE-2022-40768

In the CVE-2022-40768 problem, in the immediate beginning there are already two subsequent different "conflict actors" but both did not exist in parallel: the first *patch* (a technical institution that is to solve the problem) was proposed by the *stable* maintainer, fixing only the very issue by one additional function, but after that proposal, the *mainline* maintainer suggested a "bigger" approach to go a "cleaner way" (kernel.org, 2022d; 2022j; 2022e; 2022f; 2022g; 2022h; 2022i; 2022n; 2022a).

However, there was no objection and the *stable* maintainer immediately agreed so that the initial conflict actor has not competed with the subsequent one (ibid). In both cases, the formal actors are from the Linux Foundation, whereas other actors (one from Oracle, one private) have only tested/reviewed the two *patches*, accepting both *patches* without preferences towards any of the two, which made them part of both of the subsequent "conflict actors".

The subsequent request for splitting the *patch* (formal actor is from "acm") developed the same way (ibid): the split into two technical institutions ended up in another conflict actor, but the latter replaced the preceding one. Although this does not add much value, it is interesting that competition takes place among institutions but not among formal actors, making the latter develop one by one to different "conflict actors" whithout having different "conflict actors" in parallel. However, as already assumed before, this problem is too simple to add noteworthy value to the research.

7.2.2. CVE-2022-3435

The CVE-2022-3435 problem contains partly discussions that only focus on checking if other *sub-systems* are affected and if a self-test for such problems can be added for future issues, which makes these discussions not part of the actual "conflict" that this project is investigating. However, it can be argued that this is a factor that blurs "conflict actors" and that the argument (about this not being part of the actual "conflict") is probabilistic to some extent. However, with the argument that these discussions do not belong to the actual "conflict" in mind, the "conflict actor" development is widely comparable to the first CVE problem (kernel.org, 2022o; 2022p; 2022q; 2022r), except that it contains competition of institutions (with the institutions shaping different "conflict actors" that appear in parallel):

The original author introduced an additional check function that is comparable to another check function some lines above, which is questioned by another developer with the suggestion to merge both check functions in the first check function. However, this led to the rise of two "conflict actors", both containing one developer (*patch* developer against responsible *sub-system* maintainer). The *patch* developer focused on the simplest possible change to avoid negative impacts on *backports*, whereas the *sub-system* maintainer focused on the merge of the two check functions with the reason that this would have avoided this bug at all (which could be interpreted to imply increased resilience against future bugs: the outcome is a general check function to identify the CVE-related type and other types of problems before any code that can create problems is executed). Further, this shall increase comprehensibility. Additionally, the maintainer noted how to solve *backporting* issues.

The "conflict actor" that focused the merge of the two checks increased by another developer, which led this increased "conflict actor" to dominate and thus led to a corresponding version 2 of the *patch*, again submitted by the original *patch* developer. However, it cannot be proven if the dominant "conflict actor" that succeeded was successful because of its increased number of developers (then, 2), or if it succeeded because it contained the *sub-system* maintainer.

All three involved persons were from different formal actors. Given a mistake in the *patch* version 2, the developer decided himself to correct it and submit a version 3 without further interactions with others.

7.2.3. Adiantum

Since the preceding technologies of *adiantum* (*patches* before v3) are out of scope, the *adiantum* analysis about deriving "conflict actors" does not differ much from the two CVE problems: the mature final *patches* did not add many changes to its preceding technologies, and the latter have been already evaluated and discussed earlier outside the kernel and in the kernel's *sub-systems*, with kernel developers being involved in discussing, reviewing and testing (see the links mentioned in kernel.org, 2018). Thus, a "conflict actor" can be assumed containing Biggers and likely Crowley. But there was no longer questioning about the contained *commits* (kernel.org, 2018) at the stages of *adiantum* (v4 only adjusted v3 to fit the then-current kernel).

However, because they contained major decisions that impacted the final *adiantum* construction (Spinics, 2018a and contained follow-up links) and because the "conflict actor" derivations are already final analyses that are not capable to corrupt subsequent analyses that use its results, the previous discussions that culminated in *adiantum* shall be contained in the "conflict actor" derivation to improve its results and making it more meaningful (although the resulting *adiantum* "conflict actor" derivation can be compared to and set in contrast with other "conflict actor" derivations in this project only to a limited extent). This shall reveal more data about "conflict actor" development over time and over discussions.

In the earlier version, v2, the *Specks* cipher was used instead of *XChaCha*, and the author of *wireguard* was the one to suggest to not use a cipher that was rejected for standardization: on one hand, there was a technical institution to use the simplest construction with *Specks* that can be used on low-end devices to replace unencrypted storages and that does not add much code or complexities that need review. On the other hand, there was a technical institution to use a cipher whose design is well documented, explained/justified, and deeply analyzed and accepted by the cryptographic community, which seems to have conflicted with the other institution. These two different technical institutions led to two "conflict actors" in a common institutional environment. However, the latter institution about a cipher that is deeply analyzed and that has widespread acceptance was questioned, and the pro-*Specks* "conflict actor" has risen to three people, whereas the third person is from a different formal actor than the two *adiantum* developers.

Yet, another developer then started to argue neutrally about *Specks* but focused a theoretical but problematic alternative (which was originally put forward by the pro-*Specks* "conflict actor" to illustrate why *Specks* is suited better) and started to elaborate it (Spinics, 2018b). At this point, the "conflict actor" that was for a deeply analyzed and widely accepted cipher had already disappeared and this institution as such was no longer mentioned, although one cipher its institution referred to kept in discussion: the elaboration towards the new alternative keeps considering *ChaCha*

variants (Spinics, 2018b; 2018c). However, the pro-*Specks* "conflict actor" started to also contribute to develop the new alternative, but kept focused on comparing the elaborations to *Specks* and kept lobbying for *Specks*, but also agreed to check alternatives (Spinics, 2018c). Yet, the pro-*Specks* "conflict actor" remains but has changed its agency after the new alternative-elaborating "conflict actor" has replaced its predecessor. The formal actor that constituted the new alternative-elaborating "conflict actor" could not be identified.

Yet, subsequently, the two "conflict actors" have merged towards a collaboration to identify the eligible ciphers and constructions and combinations without a focus on pro/cons of Specks, without identifiable dissent (Spinics, 2018d and following the "next by thread" links). The institutions that make up the "conflict actor" keep developing in collaboration, and further possibilities like options that can be evaluated/chosen because of the many possibilities and different advantages and disadvantages keep being added. Additionally, further alternatives that might need more time before being ready but that might produce better compromises on the long term are put into discussion.

Nevertheless, in a later email a new entrant, a fellow of KU Leuven who says to have been in the group that rejected *Specks* in the ISO standard, revived the contra-*Specks* "conflict actor" (the one for deeply analyzed and widely accepted ciphers) by elaborating the ISO decision from a technical point of view (Spinics, 2018e). Yet, the development of the discussion had already left the realm of *Specks*, and finally found its culmination in *adiantum*.

However, in these discussions I just elaborated, it cannot be verified in this project if this competition was persistently in between institutions (= different rational arguments about how to achieve the same goal the best way, with the best institution being then adopted by all into the common institutional framework which is then to be commonly exploited by all) or if it also contained competition in between (groups of) people (= primarily, insisting on the own opinion because of biases against unforeseen alternatives, which questions own preferences). Yet, the development was in any case resolved into collaboration and finally, into a common institutional framework equally adopted and exploited by all. Here, my institutional approach has limitations because of the incapability to always and persistently prove throughout the "conflict actor" development which of the two competition types apply when: the approach can distinguish the types of competition (between people and between institutions), but not always identify when which applies in a given case. Yet, future research might improve the approach to get complementary data that can at least give comprehensible indication.

Two interesting points about the v2 discussions (within the discussions):

- 1) a discussion of opposing actors that needed to contain even pleas to remain friendly developed towards collaboration at the moment when a "conflict actor" that was against a solution without suggesting explicit alternatives was replaced by one that kept suggesting an explicit alternative. However, despite being relevant for constructive development, this general human behavior is out of scope, but this positive correlation (arguing with explicit alternative instead of focusing on rejecting what is put forward → collaborative competition about institutions with everyone discussing and arguing towards the common goal) might add an interesting perspective to the paragraph above about competition between institutions and between (groups of) people, and how to identify what takes place: this point maybe indicates a trigger for competition among institutions developing into competition between people. However, deeper elaboration of this is out of scope
- 2) KU Leuven researcher illustrates how Linux kernel discussions can spread and be found globally and throughout different groups, which gives people from other fields than Linux - but that have relevant background information - possibilities to add their perspectives (Spinics, 2018e), and to disprove biases that might had been in related Linux kernel groups. This topic and the post of this researcher has been cited several times in communities, including Linux communities, especially with reference to *Specks* that has been removed from the kernel later. An evaluation if this post was only a major reference in the discussion to remove *Specks* from the kernel or a major trajectory (or more) goes beyond the scope of this project.

Generally, the "conflict actor" approach remains interesting and it is offering new perspectives that are unbiased about "who" and "what" an actual actor is and how the actor develops - both in a comprehensible and reproducible way.

7.2.4. Wireguard & Freesync

First, *wireguard* publications are out of scope for this section. However, after an overview of the related discussions that finally made *wireguard* to end up in the kernel reveal a problem of *wireguard's* "conflict actor" derivation: it goes far beyond the capabilities of this project, even with the limitations and restrictions already introduced in the *Conflict actors & the institutional dynamics* section. Again, selections would be biased by my personal opinion about which discussions/topics to include and which not.

The interrelated discussions are mostly centered on the *wireguard*-specific mailing list on lists.zx2c4.com (https://lists.zx2c4.com/pipermail/wireguard/), but complemented by additional discussions in the related kernel.org mailing lists. Yet, given the fact that this technology has been already in wider deployment before it was merged into the kernel, a lot of discussions and topics had been open at the time that focused fixing and developing around *wireguard*. Obviously, this correlates to the previously mentioned fact that *wireguard* kept being regularly updated with further *commits* from the very time it was merged but also before.

Some points/topics in the mailing lists might be derived from kernel discussions, some not. Some might have impacted the kernel *commits* directly or indirectly, some not. A lot of technical interrelations are possible with *wireguard*: some are directly linked to Linux, some indirectly, some not at all. Determining which discussions/topics (and discussion/topic forks) to include and which not is probabilistic and would be biased: neither comprehensible nor reproducible - *wireguard's* outreach and interrelations are above the project's capabilities.

A full scale "conflict actor" derivation for *wireguard* would be itself a full project, if not more. Unfortunately, the above two paragraphs are the reason why I expect that *wireguard* would be the most indicative and most valuable "conflict actor" derivation. However, it is not realistic within this project in a manner that serves the project goals, and despite the original plan, it has to be skipped.

The "conflict actor" development of *Freesync* was limited and restricted in the *Conflict actors & the institutional dynamics* section the same way *wireguard* was limited and restricted. The remaining data of *Freesync* does not go beyond this project's capabilities, but it does not offer a "conflict actor" development with regards to the project goals: the data that remains after the limitations/restrictions I introduced is not sufficient/eligible for deriving "conflict actors" and their development. Further, potential internal discussions or meetings between the developers at AMD, which might be related to derive developing "conflict actors", are not known or reproducible: biased assumptions would be necessary to fill the gaps. Therefore, "conflict actor" derivation for *Freesync* is not capable of fulfilling its goals and thus will be skipped.

8. Public formal actors in the Linux architecture

The previously elaborated outreach of the "problem following" obviously also impacts the 4th objective: identifying appearances and relations to public formal actors that indicate how far the Linux architecture is blurred in terms of public and private entities.

Generally, the CVE pages that had appeared in this project are from Mitre, which is a private organization that is largely funded by the United States. At the same time, CVE references lead to the "National Institute of Standards and Technology", which is a public formal actor of the United States. However, both formally and informally the Linux architecture made clear to not focus and rely on CVEs, and to deploy its own institutions instead. The Linux architecture allows reporters to request and assign CVE numbers, and to keep the assigned CVE numbers documented throughout problem handling in order to keep compatibility and comprehensibility. However, it clearly does not belong to the architecture: it is only compatible, and documented if a reporter wishes.

Concerning the analyzed discussions, none contained explicit relations to public formal actors. Yet, the "conflict actor" derivation of *adiantum* related to *Specks* (although this remains out of scope for the "problem following"), and *Specks* has origins in and relations to the United States National Security Agency (kernel.org, 2018; Spinics, 2018a; 2018b; 2018c; 2018d & contained follow-up links).

Yet, the formal actors have not been immediately involved: the analyzed problem/discussion was only about using *Specks*, not developing or standardizing it, and the *Specks*-related US actors have not been involved in the analyzed problem/discussion. However, it could be argued that the research fellow (who was a member of the ISO committee that rejected *Specks* despite US officials lobbying for it) represented in this case a public formal actor. At the same time, an institution that preferred standardized cryptography formed a "conflict actor". However, the latter had a short life time and the majority of discussions (and the reason for finally replacing *Specks*) was not directly related to standardization or public formal actors but had primarily a different background.

Therefore, at least the problems that had been analyzed in this project do not contain immediate relations to public formal actors, and cannot be compared in this respect to earlier added technologies like SELinux with its public formal actor relations (the SELinux mandatory access control has been driven primarily by the US National Security Agency). In any case, the scope of the project is too small to indicate that public actors are not (or, no longer) involved in the Linux architecture: the project is just not capable to indicate any of the two possibilities.

9. Solarwinds attack: preventable by appropriate socio-technical architecture?

Again, given the interesting but unexpected development of the "problem following", the comparison with the Solarwinds case is possible only to a limited extent, mostly related to the related infrastructures. However, a related analysis has been already conducted by David Wheeler (Wheeler, 2021) and by others.

With regards to the results of this project, it can be added that the distribution among many developers makes it unlikely that an intrusion on one of the many infrastructures would have been capable to achieve such a success: many would have to have overseen such a massive anomaly.

At the same time, a lot of communities, organizations and individuals review code and test the kernel, and even more until a kernel is deployed to enterprise use cases. Any one *git* instance that *pulls* the manipulation, will output all *pulled* changes to its developer (removed + added lines): every code is likely to be reviewed by everyone who is responsible for its *sub-system* or related *sub-systems*, and by everyone whose developments depend on that code.

Complementary, a lot of standardized and individual tests are conducted to uncover as much "behaviors" as possible, and are documented and published for another review or further complementary tests as well in many cases. At the Linux operating systems in wider deployment, the kernel is transported also using a "chain" of signatures and automated *pulls* that can verify hashes, making manipulations in between problematic.

However, at the level of the Linux operating systems, the communities/organizations that develop the operating systems have to actively ensure a proper "chain" and achieve security by design also on themselves because the kernel community cannot guarantee for what occurs in its subsequent stages: there is not the one security of Linux when it comes to deployable operating systems. Thus, the users also depend on the security provision of their operating system providers, and the latter could indeed provide indepedently dangerous attack surfaces. However, major communities/organizations that provide Linux operating systems contain comparable institutions as the kernel community, and these communities/organizations belong to the Linux architecture and are involved with reciprocal feedback loops, and tracable and comprehensible "chains of trust". Yet, deploying a Linux kernel does not automatically imply this type of security, and deploying a Linux kernel does also not imply to be involved with the socio-technical Linux architecture.

Another point here is *deterrence*: an attacker cannot foresee who reviews the source code with which focus and with which capabilities, and who conducts what type of test in which environment.

Additionally, it might be added that most developers are unlikely to have open ports and attack vectors like a web server. However, even if different, it would not help an attacker to tackle the above issues.

With the "Solarwinds" type of attack in mind, it could be argued that the earlier malicious code is tried to be added, the more likely it is to be revealed given the increased number of reviews/tests that follow. With this in mind, it could be argued that such attacks are less unrealistic at the levels of Linux operating systems, although "less unrealistic" does not imply "realistic". However, again, such evaluations illustrate the importance of the operating system communities/ organizations, and when it comes to such attacks, they could be argued to be the major provider of security against them.

However, on one hand, the more important a Linux operating system is and thus, the more it is interesting for attackers, the more it is likely that its infrastructures (which in the evaluated cases equal widely the remaining distributed Linux architecture) are comparably resilient. Such attacks are made further unlikely given the feedback loops up and down when it comes especially to the widespread Linux operating systems. All that does not only decrease the likelihood of successful attacks but also increase the *deterrence*. Additionally, it has to be noted that kernel development often takes place on Linux operating systems, which creates further loops when they, e.g., test their kernels on/for their own operating system (assuming that a large number of kernel developers use Linux operating systems).

Furthermore, once a kernel is deployed to the update system of an operating system, further tests are possible given that the widespread Linux operating systems publish their data so that the kernel's "evolvement" can be analyzed and reproduced (although the latter is not necessarily "mathematically reproducible"). Indeed, verifying kernels and if they comply to what had to have happened before (and also to verify this on different hardware) is a common practice. Nevertheless, an increased risk exists at the stage of building the kernel: reproducing a very build is problematic (ibid), and the outcome cannot be always "mathematically reproduced". However, the following tests can uncover anomalies, and even if a "mathematical reproduction" at this very level is not possible, the agency/behavior of different builds can be compared (and so the data/results of tests and deployments), which further limits attack vectors and increases *deterrence*. Especially the *deterrence* is a factor that Wheeler (ibid) has neglected. The open access and possibility to review, test and harden the (build) environments and not only the code in the Linux architecture has to be considered explicitly, too.

Nevertheless, the resilience of different Linux operating systems differs: the highest degree of resilience is likely to exist on the Linux operating systems that are involved in and deployed to the kernel development and thus, most deeply involved in the different feedback loops and also to be exposed to widespread tests and reviews (of code and infrastructures) at all levels and stages. But in any case, the likelihood of such attacks can be only reduced, but it cannot be explicitly excluded, not even in the Linux architecture. Additionally, it has to be considered that such code cannot only harm in the kernel, but also in other code within a deployed operating system.

10. Assembling the architecture and concluding the project

10.1. Assembling, summing and concluding Linux's institutional architecture & dynamics

The architecture around Linux is shaped by many institutions that are not yet widespread in other organizational architectures, and that are often absent in contemporary security considerations or even contrary to existent security practices. However, in the end, the agency of the Linux architecture is the outcome of all of its institutions. Indeed, a given institution can cause a different agency in different architectures.

Complementary, the dynamics that evolve from all of Linux's institutions can in some cases make a human behavior, which can cause security-critical risks in other architectures, to an advantage that increases security and stability of the socio-technical architecture: as indicated in the "People to create compatibility between redundancy and consistency" section, the Linux architecture exploits people's (in this case, its community's) capacity to balance issues intuitively, which balances and solves inconsistencies in all the *redundancy* but also balances unintended behavior at places where unintended behavior is expected (which is the input stages).

In other architectures and situations, this intuitive human behavior can become a major attack vector, especially in situations where people can act intuitively without having an information flow that enables their "community" (or organization) to balance their behavior if their individual intuitive balancing can cause issues they are not aware of (e.g., given a lack of knowledge in all fields that are related to their action).

Despite the initial assumption that "intended input" is a key property of "secure by design" while "unintended input" can break secure by design, the Linux architecture perverted these properties with places of "expected unintended input", which implies indirectly "intended unintended input": there are several places where everyone can input to the architecture, but the Linux architecture is aware of that, expects unintended inputs and remains capable of balancing "unintended inputs" up to the levels that have occurred so far: when it comes to exploiting these places of expected unintended input for attacks, the question of what is expected is less the type of unintended input but more the amount. It is not yet known if and how it is possible to achieve denials of service by massive amounts of unintended inputs to the places where everyone can contribute: but the possibilities to simulate people with low resources and thus overwhelm the places of input could increase with artificial intelligences, but this goes beyond the capabilities of this project.

With regards to secure by design in general, it is clear that the kernel community on itself cannot ensure security by design throughout all stages up to the operating systems that are used by people and organizations: much depends on the operating system providers (communities, organizations) to keep security by design ensured up to the installed and maintained operating system.

However, the major operating system providers are both deeply intertwined and overlapping with the kernel community and thus, with the major providers, "secure by design"-related issues are likely to be revealed early: their institutions need to remain compatible to facilitate seamless interactions/exchange and omnipresent loops, on which all sides depend. Because interdependent, intertwined and overlapping, it would be immediately apparent if the institutions become incompatible and if the loops break. Yet, a problem could rise if people or organizations have no ties to the communities: they possibly cannot identify an appropriate Linux operating system that ensures a chain that keeps secure by design throughout.

A dedicated distinct/clear determination of who the "major providers" are and how "major provider" can be defined goes beyond the project scope: without having the need to focus on this distinction, this project simply assumed "major providers" to be those that are interdependent, intertwined and overlapping with the kernel community and that are involved in the mainline/stable kernels' development/maintenance and that keep feedback loops. However, with this definition, only providing feedback/reports when, e.g., bugs appear, does not differ from individual contributions and although this clearly belongs to the Linux architecture, this was not sufficient to fulfill the classification of "major providers" within this project. Obviously, determining "major providers" will be itself probabilistic if conducted by future research. The project uncovered only a few "major providers", and it does not provide an estimation how many there are (and how far they are further interrelated and overlapping with each other).

10.1.1. Competition, collaboration & dynamics

It can be argued that it was the strong competition in the fields around Linux that have facilitated and developed this capability and capacity to efficiently and effectively enable information flows and solve problems in its architecture. However, this project cannot prove but only indicate if the "collaborative competition", which could be a major

facilitator of the information flows and of the capabilities to identify and solve problems quickly (including security problems), is only a positive correlation or the outcome of Linux's "competition towards the future": currently, much competition takes place within the Linux architecture (which means that the architecture does not participate in competition but integrates it), whereas external alternatives seem to have a limited relevance at the moment.

If something can add value in future, it might be interesting to merge it into Linux, whereas Linux has no limited slots of what can be merged (the community seems to be capable to manage the suggested contributions so far). This means that contributors are not in competition with each other but only have to convince responsible developers and maintainers: competition of institutions, not of people.

Linux regularly introduces solutions which then develop to demand directly or indirectly, not only vice versa. Because Linux seems to have no competitors that could effectively "steal" its contributions (in most cases it is more competitive to contribute given the free use and reputation of Linux) while its rewards are themselves a valuable "salary", the "collaborative competition" that explicitly appreciates and reputates contributions creates incentives for developers worldwide who are thus encouraged to contribute without the need to fear the exploitation of their work without benefiting: reputation is a major institution, and it is the major reward that facilitates and stabilizes careers and salaries independent of any individual organization.

At the same time, the high legitimacy of failure (a "mistakes create experience and thus knowledge" mentality), in conjunction with the absence of seeking fault but instead to just find out how to improve or how to solve problems, mitigates deterrence against contributions. Generally, the dynamics of the architecture ensure that this "collaborative competition" is among institutions, not among people, but slips where developers fall prone to competing with each other can occur: however, such phenomenons seem to be balanced quickly within the architecture and thus dissolved back into competition among institutions, at least as far as it was observed in the project.

Deterrence of contributions also manifests in the possibility to get any code and test or play with it without the own activities being publicly known in advance (which means no pressures to find bugs or so). Beyond mitigating deterrence of contributions, the architecture provides deterrence against attackers: no one knows who tests what when where and how. An attacker has to expect anything to be identified, while the infrastructure does not contain "single points of failure" because it is distributed and heterogeneous: no one system, tool, operating system or other mean in the architecture is itself a dependency of the architecture and its products: any manipulation is highlighted on many systems at many stages and thus, likely to be identified.

The chain of trust is a major facilitator in Linux, while its *sign-off* area is a major social institution that facilitates the chain to include documentation that fosters reputation and responsibility/liability, whereas OpenPGP is the related technical institution that ensures the cryptographic security (in this case, integrity) of the chain and thus, that the chain and its contents remain resilient to manipulations.

However, although SHA1 is not yet a problem in the very use case it is deployed to, the absence of development towards best practices like SHA256 could be explained by deadlocks, that mitigate development. Although it is neither explicitly indicated nor proven, deadlocks could be linked to people that remain constantly in the same leadership positions (which could dissolve competition related to the duties of their position), such as the mainline or stable maintainers. However, the latter can be also explained by these people being competitive in tailoring to the needs of competition in the community (implying adjusting their institutions and behaviors without becoming constants: competition and collaboration remain active but satisfied by them), which would imply that this is not linked to potential deadlocks: they are only constants of the architecture (that could cause deadlocks) if they can impose their institutions over the architecture against the general demand (implying at least the majority of the community) and against the will of the community, whereas they are variables if the architecture can get rid of them once they try to do so or once they do no longer fulfill the demand against their positions. Related questions remain open, and it cannot be

foreseen at this point if the architecture will be able to manage issues like SHA1 on time if it becomes critical at some point, although it should be noted that SHA1 is already well understood and immediate breakthroughs that break SHA1 immediately for its very use case in Linux can be argued to be unlikely.

At the same time, the Minnesota issue has shown that in some circumstances developers trust, e.g., universities. Although the case has not shown vulnerabilities but indeed gave indication that a university and its people (who have to take responsibility for their actions) will not facilitate exploitations intentionally, the case might indicate a vulnerability if an attacker captures university addresses. However, so far, there is no case of exploitation, and it remains unknown if the Linux architecture has adjusted to avoid such issues in future in general. Further, it is not clear if subsequent stages would have identified the vulnerabilities because the vulnerable patches have never reached the stages after the stable kernel branches.

As already noted in conjunction with "single points of failure", redundancy is a major institution along with distribution (which is contrary to many contemporary security approaches that centralize with the goal to increase security), whereas the absence of "single points of failure" is complemented by places of "intended input" and of "expected unintended input", providing widespread socio-technical security by design: it is not always a technical institution necessary to balance a technical single point of failure or to balance a place of expected unintended input: a technical issue can be also balanced by a social institution, and the same vice versa. The redundancy and the quick balancing of inconsistencies with the ease of access of any type of information quickly but also the possibility to access the architecture at different places with different scopes/structures (e.g., lwn is a good point to start to "query" for all types of kernel related information but without operating system specific developments) forms a type of socio-technical *NoSQL* database. However, with the assumption that people practically "query" this socio-technical database through search engines, the deployed search engines can be also argued to be institutions involved in the Linux architecture, although they remain a part of the architecture just indirectly through the developers/users, just like the many tools the developers deploy to code or to test. However, beyond passive tools, for the technology *wireguard* with its major changes, an external auditing firm has been hired to cross-check the technology.

The Linux architecture and its products have no need for privacy/secrecy at most places (reports for exploitable vulnerabilities mark an exception as far as observed during this project), which shapes its agency. However, a limitation of the architecture that is itself a major institution is the "GPLv2-only" license: any code in the kernel is published under the GPLv2, explicitly not later versions of the GPL (GNU General Public License). However, this project could not observe indications that this leads to restrictions of contributions, but the project scope is not intended to evaluate that.

With a technical focus, major institutions are HTTPS (including SSL/TLS), SSH, OpenPGP, git and of course the C programming language, while Rust has just been added to the kernel after the 5 era and could develop to a major institution in future. In this respect, SHA1 and asymmetric cryptography (Curve25519, RSA, P-256) need to be mentioned as well. Important is that OpenPGP is only used for signatures, not encryption: indeed, symmetric ciphers are only part of the Linux architecture indirectly through the use of SSH and HTTPS. Of course mailing lists' email protocols that are deployed for and by git are relevant institutions as well. This is linked to much automation: on one hand, many people have to manually trigger pulls/merges, but on the other hand, the tasks themselves are automated (e.g., how to add and remove code through mails in mailing lists).

However, since it is used by many to start contributing, kernel.org and other domains are also relevant institutions, although their security relevance can be questioned given that, even if e.g. kernel.org repositories are manipulated, new contributors that use such domains to get the git repositories cannot introduce them to kernel branches, while people who derive kernels for widespread operating systems have their own existing forks where manipulations will be highlighted. Also, many developers interact with the infrastructures of, e.g. kernel.org, each day (if not hour), and get highlights of changes and of conflicts.

10.1.2. Person-centric self-organization, git and sub-systems within Linux's competition and cooperation

When it comes to the organization within Linux, maintainers and sub-systems are important institutions that shape the organizational architecture, while especially sub-systems are linked to git structures: indeed, branches, tags, repositories and their forks/clones can be argued to be themselves institutions, just like merges and pulls: merges/pulls from one stage to the next until after the final kernel-specific stages with pulls into the kernels' branches, the developers of the initial stages pull again the "overall" changes from kernel branch maintainers, which closes the loop). Additionally, the other behaviors and functions of git are institutions with corresponding impact on the architecture as well (up to behaviors like highlighting changes that contribute to other institutions, such as, e.g., to deterrence).

Generally, "pulling" defines much of the task allocations: work is put into the competition so that anyone can pick it up and suggest solutions, it is not allocated. However, when a problem is in the process of being solved within the chain of trust, the chain is seamless (everyone involved, such as a maintainer, is expected keep an eye on it) and it remains explicitly ensured that the issue remains in processing and that someone takes over.

Indeed, this might be also related to the kernel security team: someone has to ensure in security critical problems that these are processed and that someone takes over immediately. Beyond that, and although their role is limited to the initial stages of problem processing, the kernel security team also avoids that attackers can exploit vulnerabilities before the kernel community and the operating system communities/organizations can respond to the problem: no publication of an exploitable vulnerability until interim ways for mitigation can be published along with it.

Additionally, not only the documentation of code, but also the Kernel Docs (which are the foundation to get new developers) for people are important institutions: a major institution to lower the entry barriers to the Linux architecture.

With regards to ensuring that someone takes over: this reminds on another institution that is not directly related but facilitates comparable behavior at more strategic levels of the architecture: the "hand over before resign" institution ensures that maintenance remains ensured and that responsible developers do not just disappear but actively seek and wait for replacement before resigning, which is an institution that is linked to reputation. Although it is generally assumed that people keep maintaining their contributions, this becomes more informal with levels getting lower: while on higher levels, maintenance is formally allocated, on the lowest levels, it is normal that people also maintain code of others, although it is a "best practice" to keep the original authors involved if they are still active.

Based upon the Linux-specific git structures, the separation into two repositories, mainline and stable, also shape the Linux architecture, and indirectly relate to the separation in at least two major maintainers: a mainline maintainer, and at least one stable maintainer. Also indirectly related to the git repositories is the "one problem per patch" institution, that facilitates comprehensibility and simplicity. Indeed, comprehensibility and simplicity can be also seen as institutions on which Linux depends, given the massive amount of code that has to be maintained and code that is added and removed each day. Comprehensibility and simplicity also relate to the infrastructure, which is limited to simple means and to what is necessary. The persistent evolvement of one mainline kernel without starting regularly from scratch is also a major institution that shapes agency, although it can be questioned if alternatives to that would be realistic: nevertheless, "one mainline kernel" has an impact on the architecture and its dynamics that has to be considered.

The person-centric organization might remind at first glance to agile methodologies, but comparabilities and correlations should not be assumed automatically. Indeed, the focus on people and their interrelations/interactions instead of a focus on organizations (which blurs the organizations and their boundaries) could also be linked to the dynamics of regions like the Silicon Valley as analyzed and elaborated by Anna-Lee Saxenian (Saxenian, 1991; Saxenian et al., 2017). Indeed, Saxenian et al. also identified a stable balance of competition and cooperation without central governance in related regional economies (Saxenian et al., 2017, pp.148-151; Klooz, 2021, p.42):

This also links to my past thesis' elaboration of the competition-based distribution of powers (it is ensured that everything is publicly available/discussable, that it can be done/conducted by every capable entity without pre-determination or fixed allocation, and it is always the entity chosen that fits the situational demand and the "will" of the related community best, but it can and will be replaced by other entities once it abuses power or once it is no longer is competitive for other reasons) that replaced a traditional command-economy-based separation of powers in some Internet realms (including Linux, as also illustrated in several elaborations of this project) and that keeps competitive and adaptive dynamics against its demand ongoing while monopolies or abuse are stably mitigated by the given institutions (Klooz, 2021, p.44,45,51). Research like that of Saxenian is nothing new, but it has never been linked to security issues: this remains to be done by future research.

Based upon this project, it cannot be said if and how far relations exist, but it has to be noted that much Linux development has been based in the area in and around the Silicon Valley, while the Linux Foundation is itself based in San Francisco. Yet, Linux development is globally distributed and cannot be limited to any individual region or any specific economy. Institutional relations are possible but not explicitly indicated in this project. However, deeper analysis goes beyond the scope of this project.

Additionally, it has to be noted that potential types of hierarchies are, beyond the importance of reputation in interactions of people, derived from the sub-systems of the kernel, which thus have also an impact on how people are organized among and with each other: the end (the working, deployable product) and not the means (such as "departments" or specific organizational elements) determines the organization and delimitations, and the elements that are organized are people, while institutions like the "maintenance" of "sub-systems" provide order and some types of (mostly informal but applied) hierarchies. However, the latter are also derived from the end, not from the means.

Yet, the last paragraph is linked to the trust of organizations into the community and the reputations that develop within: organizations trust that developers achieve the goal in the environment of the Linux architecture without external interference. The external interference is, as far as indicated in the limited scope of this project, limited to choosing whom to support (and thus, support what the respective developer stands for). Further incentives are of course possible and likely, given that there is an employment relationship. However, given the value of reputation, which could be for well reputated kernel developers more important than their employer’s salary (because the latter is the result of the first and might be provided by alternative employers as well), the influence possibilities on the kernel developers are limited.

However, this project cannot indicate if this also applies to the other communities such as the communities of the operating systems. Nevertheless, the "Fellows" and comparable people/maintainer who have leading roles in the Linux community are representative for the absence of external influences, and the institution that is indicated by it (and by other results of the project): the self-organization of the Linux community, in which people are the constants, not organizations. However, this could be also the reason why the kernel-related problem solving is largely isolated from non-technical parts of the Linux architecture (related to the "non-intervening" institution).

Many institutions and their dynamics are formalized through the concepts "trusting the developers, not infrastructure" and "any part of the infrastructure can be compromised at any time". While major institutions are formalized, much in the community is informally handled, which is indeed not yet common in architectures that are acting in the realms of anarchy.

Of course when evaluating the Linux kernel, it has to be clear that the kernel as the architecture’s product is itself an institution, and since it is always deployed within an operating system, it should never be considered on itself: the "feedback loops" with and among different but overlapping communities and organizations are themselves important institutions. It remains probabilistic how far other means of related operating systems are seen as part of the Linux architecture. However, especially when it comes to technical initialization processes like systemd that are tailored to the Linux kernel and that strongly shape its agency while their development is interrelated and interacting with the kernel development, it becomes clear that the Linux architecture does not just output a kernel but also other software

that goes along with it: a Linux operating system cannot run without an initalization process, and especially with systemd-based Linux systems (which is the majority of Linux operating systems), this process is deeply involved in managing the kernel and other parts of the operating system throughout runtime. Also, security and other issues can also rise from compatibility and interactions with hardware.

10.1.3. Implicit and indirect institutions and dynamics

The "input possibilities" and the potential places of contribution and of reporting/filing in the Linux architecture are mostly informally separated without distinct boundaries/allocations about "what to input/file/contribute/report where": the "input possibilities" of the Linux architecture can be summed up to "prefer risking unintended input over hindering intended input" from people, which leads unintended inputs at related places of input to be expected.

Informal separations without distinct boundaries/allocations are also a common institution in the mailing lists and elsewhere: boundaries among mailing lists are flexible and always derived from the end that is to be achieved, not from the means (which was already noted in the last section). This can lead to blurs and overlaps, just like in the "input possibilities". However, this is intended and expected in the architecture.

However, institutions, which ensure developers' awareness of (and means to identify) what is beyond the currently processed problem's boundary/allocation and what has thus to be considered to achieve the current goal, were not explicitly revealed. However, the organization in solving problems and the dynamics cannot be explained without such an awareness and related means to identify what has to be considered beyond the apparent relations of the very problem (such as in `1c08a1` that was adjusted to fit *adiantum*, although it was not explicitly related to it), and even if not explicitly indicated in this project's scope, git could be linked because it offers possibilities to output such information and it is available to and used by anyone involved in an issue: this would imply that code determines the allocations/boundaries in given issues/problem solving in given situations. If the code changes, allocations/boundaries can do the same. Beyond git, widespread testing and related activities might reveal further interactions and interrelations that are then to be considered and that might impact considered allocations/boundaries.

Other institutions that are important for the Linux architecture are the possibilities to fork (e.g., forks of any kernel but also of operating systems), which also mitigates abuse within the kernel community because the community as a whole could be replaced or split if a capable amount of people perceives this to be necessary. This also imposes limitations to decisions that are made by developers without justifying what they do: this is respected and accepted in some circumstances, if not even expected, but even maintainer of kernel branches need to keep others convinced to "follow" their conduct. These institutions are linked to the public availability of all data and information (and the ease to get and use all related data/information: code/products, decisions, discussions, justifications), whereas this public availability is itself an important institution and with regards to fork possibilities and abuse mitigation, it is also a major facilitator.

All that contributes to the dynamics that form the competition-based "distribution of powers" that enables the community to informally act flexibly and tailor efficiently and quickly to its demand, with limited risks of monopolies or abuse as long as the dynamics are balanced (this can be argued to be an equilibria). However, breaking these dynamics is unlikely to happen as long as data/information is publicly available and usable, whereas public conduct seems so far to be more competitive than alternatives, which makes alternatives not likely to succeed. Yet, with the limited scope of this project, indication is limited, especially when it comes to long term developments. Deeper analyses goes beyond the scope of this project, but it is possible that the competition-based "distribution of powers" in conjunction with competition of institutions (instead of competition of people) are facilitators of the stable balance of competition and collaboration that Saxenian et al. (2017) elaborated.

As elaborated earlier, because of its major product (the kernel and its special circumstances and properties) and the different privacy needs, the Linux architecture is not automatically a model for all fields/realms, although the development of "what is privacy" might make the Linux architecture in future attractive also to fields/realms for which it is not interesting as of today.

With regards to security and dedicated security, it is a matter of perspective if this is about the existence or the absence of an institution: it can be argued that the Linux architecture has an institution that avoids the existence of dedicated security considerations except the stage of the kernel security team and maybe some people that contribute to managing infrastructures like that of kernel.org (the latter could not be explicitly evaluated in this project). But it could also be argued that the Linux architecture, unlike other architectures, lacks an institution that involves dedicated security considerations except the noted exceptions. However, in the end, with regards to the project goals, it is important to note that in general, the Linux architecture solves problems generically - it does generally not consider security as something dedicated, but as something integrated. If it is not secure, it is a problem, and a problem has to be solved. Thus, problems need to be solved securely. This argumentation chain is not an explicit institution in the architecture but only one potential explanation of the underlying culture that is solving problems in Linux.

Indeed, if security is dedicated, it means that security considerations could bypass other considerations (which are not the focus of, e.g., security personnel), and thus, decrease, e.g., productivity or undermine affected sub-systems, organizational elements or goals (=reducing competitive advantage in affected fields), which could create incentives for the affected entities to get rid of the imposed security means (formally or informally, technically or socially). The outcome is practically not an increase of security but a shift of security: e.g., increasing security of confidentiality but decreasing the security of achieving/keeping competitive advantage (e.g., due to higher costs and less usability/ productivity of systems, which could be linked indirectly to security of availability), which is the major facilitator of any security within an organization (there is not just information security but many types of security).

Vice versa, if security is not integrated, other considerations are more likely to cause security issues because they do not involve security considerations: separating security and other considerations can only work if there is an omnipresent and persistent perfect information flow throughout. When it comes to code, security is often integrated into institutions like "simplicity" or "one patch per problem", which maximize in conjunction with massive review/ testing and public and widespread contributions that issues (= problems) are found and solved quickly, including issues that cause information security problems. Nevertheless, it has to be clearly said that, with the last two paragraphs in mind, "what is security" is as artificial and probabilistic as "what is private": everything is linked to some type of security, and finally, from a radical point of view, it can be argued that it is always only about problems that have to be solved, and any separation of security splits holistic problem solving and undermines information flows.

With security being integrated in Linux, the flexible/adaptive goal-oriented and goal-structured organization where the constants are the competitive product (in this case, code and its evolvement) and the competition of institutions from which everything else is derived from and tailored to, ensures the rest. Linux trusts its competition. However, the question for future projects is how to achieve such an intended competition that achieves the intended goals within a given field / for given goals.

However, the limited dedication of security such as in the kernel security team links back to privacy: indeed, the need to keep exploitable flaws confidential for some time can be seen as a type of privacy. Thus, with this argumentation in mind, organizations that have more privacy needs are likely to have more needs for dedicated security: in the Linux architecture, dedicated security seems to exist where types of privacy are relevant (this is primarily when exploitable flaws are processed before interim mitigation possibilities are known), whereas in other fields in Linux, security is integrated. With this correlation in mind, it could be argued that the need for dedicated security will decrease with decreasing privacy needs. As elaborated earlier, at this point, it cannot be excluded that societies and their organizations have already begun to adjust this: making things public that used to be private.

I consider the conflict actor development within its chapter sufficiently comprehensible and coherent so that a dedicated summary is not necessary. However, with the above summary of the Linux architecture in mind, it could be worth to review the related chapter "Conflict actors & the institutional dynamics" (especially the extended *adiantum* conflict actor elaboration) at this point with the recent elaboration/summary in mind: especially points that are related to competition might shed a new light on the conflict actor development.

10.2. Concluding the approach & outreach

For the reasons elaborated in the introduction, my problem following approach was intended to uncover the Linux architecture without biases, and map where and how problems are solved within. Most approaches as of today tend to impose biases which can corrupt the results, and tend to impose false assumptions: misleading false assumptions and biases can be already introduced just by imposing structures (unfortunately, avoiding misleading/unproven/corrupting assumptions/biases/structures also leads the approach to create decreased readability in the report). My approach has been successful, and at some places, it even revealed false assumptions I imposed when I designed/elaborated the objectives in the planning phase. However, this also leads to the problems of the approach: the originally unexpected isolated self-organization around technical problem solving hindered the objectives that followed the second objective.

The absence of biases also makes it impossible to foresee how the project develops: does it develop into 50 pages of research of 500? Indeed, any project has limitations of capacities, and this leads inevitably to one bias which has to be introduced: at some places, I had to decide which of the many interrelations are relevant and which not. Indeed, a radical implementation of my approach without limiting the recursions of interrelations, would uncover the whole connected "world". It can be questioned if limiting the approach to the capacities of the project corrupts the results, but formally, it is a bias and has to be noted as such. However, the approach remains comprehensible and reproducible, which also applies to the limiting of the approach to the project capacities: it is distinguishable when recursions are skipped to not exceed the project capacities, and thus, the data and the results can be evaluated in light of the known, comprehensible/reproducible limitations. With this in mind, it has to be noted that any delimitation of the "Linux architecture" is itself a probabilistic phenomenon. However, this fact has been elaborated sufficiently during the project.

My approach of deriving conflict actors in an equally unbiased and reproducible/comprehensible way reveals new (comprehensible and reproducible) perspectives on how security- and stability-providing competition and collaboration, where institutions instead of people are the major convergence of competition, evolve and take place. However, this project has had no capacity to fully reveal how its environment, in which all that takes place, has evolved. The latter remains an important question for future research.

However, another important derivation is that different institutions and behaviors can have different impacts in different architectures: institutions and behaviors that prove disadvantageous in one architecture can prove advantageous in another. Thus, behavior has always to be considered in its environment.

Just like in my last thesis, security by design has proven to be a reliable constant in evaluating socio-technical architectures. Indeed, once socially and technically unintended inputs are "expected" or "absent", and once any potential flaw can be balanced at another place (avoid single points of failure), a system is hard to break, even if single flaws appear: indeed, flaws will occur in any system, but this does not make them automatically to (critical) security risks. However, it has to be clear that the theoretically perfect and omnipresent implementation/application of security by design can hardly be imposed 100% on any architecture. And even the Linux architecture is not provably secure in an all-encompassing way: there are still open questions about how and where the Linux architecture is vulnerable to what type of attacks.

Indeed, the project had a limited scope that analyzed just small parts of the architecture, which implies that much is more indicative than proven as long as the Linux architecture, its environment, the "actively and intentionally developed anarchy" and their interactions, overlaps, outreaches and relations are not better understood. However, this was clear from the very beginning and was intended. As far as it concerns explicit vulnerability indications, open questions exist when it comes to the use of SHA1 and the use of seemingly trustworthy email addresses (e.g., from universities). Additionally, existing criticism of the Linux architecture about occurring behaviors (offensive/insulting/polarizing behaviors) among people needs elaboration, and has to be set in contrast to the architecture to identify with

what it is related: it is not known if this topic is independent or related with other parts of the architecture and with security-relevant topics - biases from traditional architectures in traditional social anarchy have to be avoided and thus, nothing should be taken for granted.

Another issue that was not analyzed given the isolation of the analyzed problems is funding/spending: indeed, fundings and spendings can create strong incentives in any architecture and for any competition and collaboration. However, especially during the problem solving, several recursions and relations/links had to be skipped to both avoid the project exceeding its capacities and to avoid the project petering out. Given the architecture as far as it is known so far, it is unlikely that any recursion/relation/link had immediately led to funding/spending related topics. Yet, any of the skipped recursions/relations/links is worth to be analyzed to finally create a holistic picture of the Linux architecture. But especially analysis from the non-technical elements of the architecture could add complementary data because the isolated self-organization of technical problem solving has limited relations to fundings/spendings.

Furthermore, future research might consider that there are stages before the implementation of code, and on one hand, they share many institutions with the Linux architecture (and can from some perspectives be argued to be part of the architecture) and on the other hand, they are the preceding stages whose output is sometimes the input in Linux: most notably, the socio-technical architectures that resulted in the Advanced Encryption Standard, SHA-2, SHA-3 and Argon2 (the last deployed the institutions of the first, even if not organizationally related). Indeed, it could be argued that it is not AES, SHA-2, SHA-3 and Argon2 that have not been broken despite the oldest being over 20 years old, but that instead it is the original Advanced Encryption Standardization and Selection Process (and its social and technical institutions and all its related and potentially overlapping entities that had been involved) which has not been broken: the algorithms that have evolved this way remain resilient and unbroken.

This project offers much indication and it revealed several perspectives that break with existing but questionable assumptions. It can be argued that its major value is to indicate what existing presumptions have to be questioned and what has to be researched next in order to better understand existing security provisions and to learn how to provide security where it does not yet exist, and maybe, if applicable, to learn how to integrate security instead of dedicating it.

Bibliography

Alshaikh, M., 2020. Developing cybersecurity culture to influence employee behavior: A practice perspective. Computers & Security, 98. [doi] Available at: https://doi.org/10.1016/j.cose.2020.102003 [Accessed 31 March 2023].

Amazon, 2022. Amazon EC2 FAQs. [online] Available at: https://aws.amazon.com/ec2/faqs/ [Accessed 31 March 2023].

Android Open Source Project, 2022a. Content License. [online] Available at: https://source.android.com/license [Accessed 13 November 2022].

Android Open Source Project, 2022b. Legal Notice. [online] Available at: https://source.android.com/legal [Accessed 13 November 2022].

Android Open Source Project, 2022c. Kernel overview. [online] Available at: https://source.android.com/docs/core/architecture/kernel [Accessed 13 November 2022].

Android Open Source Project, 2022d. Stable Kernel Releases & Updates. [online] Available at: https://source.android.com/docs/core/architecture/kernel/releases [Accessed 13 November 2022].

Arista, 2023. The World's Most Advanced Network Operating System. https://www.arista.com/en/products/eos [Accessed 30 March 2023].

Biggers, E., 2018. [PATCH v4 14/14] crypto: adiantum - add Adiantum support. [online] Available at: https://lwn.net/ml/linux-kernel/20181117012631.23528-15-ebiggers@kernel.org/ [Accessed 31 March 2023].

Brass, I., & Sowell, J. H., 2021. Adaptive governance for the Internet of Things: Coping with emerging security risks. Regulation & Governance, 15(4), pp.1092-1110. [doi] Available at: https://doi.org/10.1111/rego.12343 [Accessed 31 March 2023].

Broz, Milan, 2020. DMCrypt. [online] Available at: https://gitlab.com/cryptsetup/cryptsetup/-/wikis/DMCrypt [Accessed 01 March 2023].

Broz, Milan, 2022. LUKS2-docs. [online] Available at: https://gitlab.com/cryptsetup/LUKS2-docs [Accessed 01 March 2023].

Broz, Milan, 2023b. cryptsetup. [online] Available at: https://gitlab.com/cryptsetup/cryptsetup [Accessed 01 March 2023].

Callas, J., Donnerhacke, L., Finney, H., Shaw, D., & Thayer, R., 2007. RFC 4880: OpenPGP Message Format. [online] Available at: https://datatracker.ietf.org/doc/html/rfc4880 [Accessed 31 March 2023].

Castiglione, D., 2015. Introduction the Logic of Social Cooperation for Mutual Advantage – The Democratic Contract. Political Studies Review, 13(2), pp.161–175. [doi] Available at: https://doi.org/10.1111/1478-9302.12080 [Accessed 31 March 2023].

Chinkin, C., & Kaldor, M., 2017. International law and new wars. Cambridge: Cambridge University Press. [doi] Available at: https://doi.org/10.1017/9781316759868 [Accessed 31 March 2023].

Cisco, 2021a. Cisco IOS XE Programmability. [pdf] Cisco Systems, Inc. Available at: https://www.cisco.com/c/dam/en/us/products/collateral/enterprise-networks/nb-06-ios-xe-prog-ebook-cte-en.pdf [Accessed 30 March 2023].

Cisco, 2021b. IOS XR7 Data Sheet. [online] Cisco Systems, Inc. Available at: <https://www.cisco.com/c/en/us/products/collateral/ios-nx-os-software/ios-xr-software/ datasheet-c78-743014.html> [Accessed 24 September 2021; link broken].

Cisco, 2023a. IOS XR7 Data Sheet. https://www.cisco.com/c/en/us/products/collateral/ios-nx-os-software/ios-xr-software/datasheet-c78-743014.html [Accessed 30 March 2023].

Cisco, 2023b. Command Reference, Cisco IOS XE Everest 16.6.x (Catalyst 9400 Switches). https://www.cisco.com/c/en/us/td/docs/switches/lan/catalyst9400/software/release/16-6/command_reference/b_166_9400_cr/b_166_9400_cr_chapter_010.html [Accessed 30 March 2023].

Cisco, 2023c. Introduction: Cisco NX-OS. https://developer.cisco.com/docs/nx-os/#!open-nx-os-and-linux-introduction [Accessed 30 March 2023].

Cook, K., 2021. A statement on the UMN mess. [online] Available at: https://lwn.net/Articles/854064/ [Accessed 31 March 2023].

Corbet, J., 2022. Whatever happened to SHA-256 support in Git?. [online] Available at: https://lwn.net/Articles/898522/ [Accessed 31 March 2023].

Crowley, P., & Biggers, E., 2018a. Adiantum and HPolyC. [online] Available at: https://github.com/google/adiantum [Accessed 31 March 2023].

Crowley, P., & Biggers, E., 2018b. Adiantum: length-preserving encryption for entry-level processors. [doi] Available at: https://doi.org/10.13154/tosc.v2018.i4.39-61 [Accessed 31 March 2023].

Debian, 2022a. Reasons to use Debian. [online] Available at: https://www.debian.org/intro/why_debian [Accessed 31 March 2023].

Debian, 2022b. Our Philosophy: Why we do it and how we do it. [online] Available at: https://www.debian.org/intro/philosophy [Accessed 31 March 2023].

Debian, 2022c. CVE-2022-3435. [online] Available at: https://security-tracker.debian.org/tracker/CVE-2022-3435 [Accessed 31 March 2023].

Donenfeld, J. A., & Milner, K., 2017. Formal verification of the WireGuard protocol. Technical Report. [online] Available at: https://www.wireguard.com/papers/wireguard-formal-verification.pdf [Accessed 30 March 2023].

Donenfeld, J. A., 2017. Wireguard: next generation kernel network tunnel. In: Network and Distributed System Security (NDSS) Symposium. pp. 1-12. Internet Society. [pdf] Available at: https://www.ndss-symposium.org/wp-content/uploads/2017/09/ndss2017_04A-3_Donenfeld_paper.pdf [Accessed 30 March 2023].

Fedora, 2022a. [SECURITY] Fedora 35 Update: kernel-5.19.15-101.fc35. [online] Available at: https://lists.fedoraproject.org/archives/list/package-announce@lists.fedoraproject.org/message/VNN3VFQPECS6D4PS6ZWD7AFXTOSJDSSR/ [Accessed 31 March 2023].

Fedora, 2022a. Fedora's Mission and Foundations. [online] Available at: https://docs.fedoraproject.org/en-US/project/ [Accessed 31 March 2023].

Fedora, 2022a. kernel-5.19.15-101.fc35. [online] Available at: https://bodhi.fedoraproject.org/updates/FEDORA-2022-b948fc3cfb [Accessed 31 March 2023].

Fedora, 2022b. kernel-5.19.15-201.fc36. [online] Available at: https://bodhi.fedoraproject.org/updates/FEDORA-2022-2cfbe17910 [Accessed 31 March 2023].

Fedora, 2022c. kernel-5.19.15-301.fc37. [online] Available at: https://bodhi.fedoraproject.org/updates/FEDORA-2022-1a5b125ac6 [Accessed 31 March 2023].

Fedora, 2022d. QA:Testcase kernel regression. [online] Available at: https://fedoraproject.org/wiki/QA:Testcase_kernel_regression [Accessed 31 March 2023].

Fedora, 2022e. kernel-tests. [online] Available at: <https://pagure.io/kernel-tests > [Accessed 31 March 2023].

Fedora, 2022f. [SECURITY] Fedora 35 Update: kernel-5.19.15-101.fc35. [online] Available at: https://lists.fedoraproject.org/archives/list/package-announce@lists.fedoraproject.org/message/VNN3VFQPECS6D4PS6ZWD7AFXTOSJDSSR/ [Accessed 31 March 2023].

Fedora, 2022g. [SECURITY] Fedora 36 Update: kernel-5.19.15-201.fc36. [online] Available at: https://lists.fedoraproject.org/archives/list/package-announce@lists.fedoraproject.org/message/GGHENNMLCWIQV2LLA56BJNFIUZ7WB4IY/ [Accessed 31 March 2023].

Fedora, 2022h. [SECURITY] Fedora 37 Update: kernel-5.19.15-301.fc37. [online] Available at: https://lists.fedoraproject.org/archives/list/package-announce@lists.fedoraproject.org/message/S2KTU5LFZNQS7YNGE56MT46VHMXL3DD2/ [Accessed 31 March 2023].

Fedora, 2022i. Fedora alert FEDORA-2022-b948fc3cfb (kernel). [online] Available at: https://lwn.net/Articles/911439/ [Accessed 31 March 2023].

Fedora, 2022j. Fedora alert FEDORA-2022-2cfbe17910 (kernel). [online] Available at: https://lwn.net/Articles/911140/ [Accessed 31 March 2023].

Fedora, 2022k. Fedora alert FEDORA-2022-1a5b125ac6 (kernel). [online] Available at: https://lwn.net/Articles/911549/ [Accessed 31 March 2023].

Fedora, 2022l. Bug 2128462 (CVE-2022-40768) - CVE-2022-40768 kernel: leak of sensitive information due to uninitialized data in stex_queuecommand_lck() in drivers/scsi/stex.c. [online] Available at: https://bugzilla.redhat.com/show_bug.cgi?id=2128462 [Accessed 31 March 2023].

Fedora, 2022m. Bug 2128463 - CVE-2022-40768 kernel: leak of sensitive information due to uninitialized data in stex_queuecommand_lck() in drivers/scsi/stex.c [fedora-all]. [online] Available at: https://bugzilla.redhat.com/show_bug.cgi?id=2128463 [Accessed 31 March 2023].

Fedora, 2022n. kernel-5.19.15-201.fc36 . [online] Available at: https://bodhi.fedoraproject.org/updates/FEDORA-2022-2cfbe17910 [Accessed 31 March 2023].

Fedora, 2022o. Updates. [online] Available at: https://bodhi.fedoraproject.org/updates/?packages=kernel&page=8 [Accessed 31 March 2023].

Fedora, 2022p. Re: Hardware without AES-NI: use xchacha12/Adiantum instead of AES-XTS. [online] Available at: https://lists.fedoraproject.org/archives/list/devel@lists.fedoraproject.org/message/PLHDMYFYENKKY4UUTBTFLJBGISF5ZXJU/ [Accessed 31 March 2023].

Fogg, B. J., 2009. A behavior model for persuasive design. In: Proceedings of the 4th international Conference on Persuasive Technology, pp. 1-7. Association for Computing Machinery. [doi] Available at: https://doi.org/10.1145/1541948.1541999 [Accessed 31 March 2023].

Foley, M.J., 2019. Microsoft's Chromium-based Edge browser to be generally available January 15, 2020 . [online] Available at: https://www.zdnet.com/article/microsofts-chromium-based-edge-browser-to-be-generally-available-january-15-2020/ [Accessed 31 March 2023].

Georgiadou, A., Mouzakitis, S., Bounas, K., & Askounis, D., 2020. A cyber-security culture framework for assessing organization readiness. Journal of Computer Information Systems, 62(3), pp.452-462. [doi] Available at: https://doi.org/10.1080/08874417.2020.1845583 [Accessed 31 March 2023].

GitHub, 2022. torvalds / linux. [online] Available at: https://github.com/torvalds/linux [Accessed 31 March 2023].

GnuPG Project, 2014. What's new in GnuPG 2.1. [online] Available at: https://www.gnupg.org/faq/whats-new-in-2.1.html [Accessed 31 March 2023].

Google, 2015. Do the ChaCha: better mobile performance with cryptography. [online] Available at: https://blog.cloudflare.com/do-the-chacha-better-mobile-performance-with-cryptography/ [Accessed 03 March 2023].

Google, 2019. Introducing Adiantum: Encryption for the Next Billion Users. [online] Available at: https://security.googleblog.com/2019/02/introducing-adiantum-encryption-for.html [Accessed 31 March 2023].

Google, 2022. Google Infrastructure Security Design Overview. [pdf] Available at: https://cloud.google.com/docs/security/infrastructure/design/resources/google_infrastructure_whitepaper_fa.pdf [Accessed 31 March 2023].

Hoofnagle, C.J., King, J., Li, S. & Turow, J., 2010. How Different are Young Adults from Older Adults When it Comes to Information Privacy Attitudes and Policies?. SSRN Electronic Journal. [doi] Available at: http://dx.doi.org/10.2139/ssrn.1589864 [Accessed 31 March 2023].

Huntington, S. P., 1968. Political order in changing societies. New Haven: Yale University Press.

IDC, 2020. IDC's Worldwide Quarterly Ethernet Switch and Router Trackers Show Mixed Results in Fourth Quarter of 2020. [online] IDC Corporate USA. Available at: https://www.idc.com/getdoc.jsp?containerId=prUS47525621 [Accessed 19 September 2021; link broken].

IDC, 2022. IDC's Worldwide Quarterly Ethernet Switch and Router Trackers Show Continued Growth in Third Quarter of 2022. [online] IDC Corporate USA. Available at: https://www.idc.com/getdoc.jsp?containerId=prUS49948322 [Accessed 30 March 2023].

A. C. S. Santos, K. Tarrit and M. Mirakhorli, 2017. A Catalog of Security Architecture Weaknesses. In: IEEE (Institute of Electrical and Electronics Engineers), 2017 IEEE International Conference on Software Architecture Workshops (ICSAW). Gothenburg, Sweden, 5-7 April 2017. New York: IEEE Xplore. pp.220-223. [doi] Available at: http://dx.doi.org/10.1109/ICSAW.2017.25 [Accessed 31 March 2023].

Jivsov, A., 2012. RFC 6637: Elliptic curve cryptography (ECC) in OpenPGP. [online] Available at: https://datatracker.ietf.org/doc/html/rfc6637 [Accessed 31 March 2023].

Juniper, 2021. Junos OS Evolved Overview. [online] Juniper Networks, Inc. Available at: https://www.juniper.net/documentation/us/en/software/junos/evo-overview/topics/concept/evo-overview.html [Accessed 30 March 2023].

Kernel Docs, 2022a. The Linux Kernel Documentation 5.19: HOWTO do Linux kernel development. [online] Available at: https://www.kernel.org/doc/html/v5.19/process/howto.html [Accessed 31 March 2023].

Kernel Docs, 2022b. The Linux Kernel Documentation 5.19: Kernel Maintainer PGP guide. [online] Available at: https://www.kernel.org/doc/html/v5.19/process/maintainer-pgp-guide.html [Accessed 31 March 2023].

Kernel Docs, 2022c. The Linux Kernel Documentation 5.19: The tip tree handbook. [online] Available at: https://www.kernel.org/doc/html/v5.19/process/maintainer-tip.html [Accessed 31 March 2023].

Kernel Docs, 2022d. The Linux Kernel Documentation 5.19: Submitting patches: the essential guide to getting your code into the kernel. [online] Available at: https://www.kernel.org/doc/html/v5.19/process/submitting-patches.html [Accessed 31 March 2023].

Kernel Docs, 2022e. The Linux Kernel Documentation 5.19: Creating Pull Requests. [online] Available at: https://www.kernel.org/doc/html/v5.19/maintainer/pull-requests.html [Accessed 31 March 2023].

Kernel Docs, 2022f. The Linux Kernel Documentation 5.19: Everything you ever wanted to now about Linux-stable releases. [online] Available at: https://www.kernel.org/doc/html/v5.19/process/stable-kernel-rules.html [Accessed 31 March 2023].

kernel.org, 2018. [PATCH v4 00/14] crypto: Adiantum support. [online] Available at: https://lore.kernel.org/lkml/20181117012631.23528-1-ebiggers@kernel.org/ [Accessed 31 March 2023].

kernel.org, 2019. ChangeLog-5.0. [online] Available at: https://cdn.kernel.org/pub/linux/kernel/v5.x/ChangeLog-5.0 [Accessed 30 March 2023].

kernel.org, 2020a. ChangeLog-5.6. [online] Available at: https://cdn.kernel.org/pub/linux/kernel/v5.x/ChangeLog-5.6 [Accessed 30 March 2023].

kernel.org, 2020b. Security bugs. [online] Available at: https://www.kernel.org/doc/html/latest/_sources/admin-guide/security-bugs.rst.txt [Accessed 30 March 2023].

kernel.org, 2021. Report on University of Minnesota Breach-of-Trust Incident. [online] Available at: https://lore.kernel.org/lkml/202105051005.49BFABCE@keescook/ [Accessed 31 March 2023].

kernel.org, 2022a. ChangeLog-5.19.16. [online] Available at: https://cdn.kernel.org/pub/linux/kernel/v5.x/ChangeLog-5.19.16 [Accessed 31 March 2023].

kernel.org, 2022b. New CVE entries this week. [online] Available at: R9aV9JKqZQ@mail.gmail.com (https://lore.kernel.org/all/CAODzB9q+vbdYe_OOuyo2+DOQQsm3x-9M7zxpEd=-<a href=)/T/" class="bare">https://lore.kernel.org/all/CAODzB9q+vbdYe_OOuyo2+DOQQsm3x-9M7zxpEd=-R9aV9JKqZQ@mail.gmail.com/T/ [Accessed 31 March 2023].

kernel.org, 2022c. The Linux Kernel Archives. [online] Available at: https://kernel.org/ [Accessed 31 March 2023].

kernel.org, 2022d. [PATCH] scsi: stex: properly zero out the passthrough command structure. [online] Available at: https://lore.kernel.org/all/20220908145154.2284098-1-gregkh@linuxfoundation.org/ [Accessed 31 March 2023].

kernel.org, 2022e. [PATCH v2] scsi: stex: properly zero out the passthrough command structure. [online] Available at: https://lore.kernel.org/all/YxrjN3OOw2HHl9tx@kroah.com/ [Accessed 31 March 2023].

kernel.org, 2022f. Re: [PATCH v2] scsi: stex: properly zero out the passthrough command structure. [online] Available at: https://lore.kernel.org/all/fd6a94cd-6d71-f241-fc7b-d8613c1c2616@acm.org/ [Accessed 31 March 2023].

kernel.org, 2022g. Re: [PATCH v2] scsi: stex: properly zero out the passthrough command structure. [online] Available at: https://lore.kernel.org/all/6ee86af6-6a0d-4c8d-439d-0ea58dc7c743@suse.com/ [Accessed 31 March 2023].

kernel.org, 2022h. Re: [PATCH v2] scsi: stex: properly zero out the passthrough command structure. [online] Available at: https://lore.kernel.org/all/YzHQllbrv5zxexYD@kroah.com/ [Accessed 31 March 2023].

kernel.org, 2022i. [PATCH 5.19 12/33] scsi: stex: Properly zero out the passthrough command structure. [online] Available at: https://lore.kernel.org/lkml/20221013175145.670977674@linuxfoundation.org/ [Accessed 31 March 2023].

kernel.org, 2022j. PATCH] scsi: stex: properly zero out the passthrough command structure. [online] Available at: https://lore.kernel.org/all/CALV6CNMg0PGYteL5xP-rYQup1MjxVFPogMUH6ULwwvqeKvUzxA@mail.gmail.com/t/#m004a4aead3f12418141f7bc04fff202ec31c128d [Accessed 31 March 2023].

kernel.org, 2022k. ChangeLog-5.10.148. [online] Available at: https://cdn.kernel.org/pub/linux/kernel/v5.x/ChangeLog-5.10.148 [Accessed 31 March 2023].

kernel.org, 2022l. ChangeLog-5.15.74. [online] Available at: https://cdn.kernel.org/pub/linux/kernel/v5.x/ChangeLog-5.15.74 [Accessed 31 March 2023].

kernel.org, 2022m. ChangeLog-5.4.218. [online] Available at: https://cdn.kernel.org/pub/linux/kernel/v5.x/ChangeLog-5.4.218 [Accessed 31 March 2023].

kernel.org, 2022n. [PATCH 5.19 00/33] 5.19.16-rc1 review. [online] Available at: https://lore.kernel.org/all/20221013175145.236739253@linuxfoundation.org/ [Accessed 31 March 2023].

kernel.org, 2022o. [PATCH net] ipv4: Handle attempt to delete multipath route when fib_info contains an nh reference. [online] Available at: https://lore.kernel.org/netdev/20221005181257.8897-1-dsahern@kernel.org/T/#u [Accessed 31 March 2023].

kernel.org, 2022p. [PATCH v2 net] ipv4: Handle attempt to delete multipath route when fib_info contains an nh reference. [online] Available at: https://lore.kernel.org/netdev/73a235e9-8a9c-a212-719f-15527de359fb@kernel.org/T/ [Accessed 31 March 2023].

kernel.org, 2022q. [PATCH v3 net] ipv4: Handle attempt to delete multipath route when fib_info contains an nh reference. [online] Available at: https://lore.kernel.org/netdev/166512901590.847.15819501464797229185.git-patchwork-notify@kernel.org/T/#t [Accessed 31 March 2023].

kernel.org, 2022r. [PATCH net 11/14] netfilter: nf_tables: disallow binding to already bound chain. [online] Available at: https://lore.kernel.org/all/20220824220330.64283-12-pablo@netfilter.org/ [Accessed 31 March 2023].

kernel.org, 2022s. New CVE entries this week. [online] Available at: g@mail.gmail.com (https://lore.kernel.org/all/CAODzB9rW7E3W75GsNMZPqL5O1a_6tfZHALcLeYcSwwTA-ZVd=<a href=)/t/" class="bare">https://lore.kernel.org/all/CAODzB9rW7E3W75GsNMZPqL5O1a_6tfZHALcLeYcSwwTA-ZVd=g@mail.gmail.com/t/ [Accessed 31 March 2023].

kernel.org, 2022t. ChangeLog-5.19.16. [online] Available at: https://cdn.kernel.org/pub/linux/kernel/v5.x/ChangeLog-5.19.16 [Accessed 31 March 2023].

kernel.org, 2022u. ChangeLog-5.19.17. [online] Available at: https://cdn.kernel.org/pub/linux/kernel/v5.x/ChangeLog-5.19.17 [Accessed 31 March 2023].

kernel.org, 2022v. ChangeLog-6.0. [online] Available at: https://cdn.kernel.org/pub/linux/kernel/v6.x/ChangeLog-6.0 [Accessed 31 March 2023].

kernel.org, 2022w. ChangeLog-5.10.158. [online] Available at: https://cdn.kernel.org/pub/linux/kernel/v5.x/ChangeLog-5.10.158 [Accessed 31 March 2023].

kernel.org, 2022x. dm-crypt mailing list. [online] Available at: https://lore.kernel.org/dm-crypt/ [Accessed 31 March 2023].

kernel.org, 2022y. dm-crypt mailing list, search "adiantum". [online] Available at: https://lore.kernel.org/dm-crypt/?q=adiantum [Accessed 31 March 2023].

kernel.org, 2023a. a635a38ef9405fdfcfe97f3a435393c1e9cae971. [online] Available at: https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/tree/COPYING [Accessed 31 March 2023].

kernel.org, 2023b. 2. How the development process works. [online] Available at: https://www.kernel.org/doc/html/latest/process/2.Process.html?highlight=merge%20window [Accessed 31 March 2023].

kernel.org, 2023c. cryptsetup mailing list. [online] Available at: https://lore.kernel.org/cryptsetup/ [Accessed 31 March 2023].

kernel.org, 2023d. cryptsetup mailing list. [online] Available at: https://lore.kernel.org/cryptsetup [Accessed 31 March 2023].

Klooz, 2020. How to Hack Engineers: An “Unknown Knowledge Attack”. [online] Available at: https://dzone.com/articles/the-agency-of-people-or-how-to-hack-engineers-not [Accessed 30 March 2023].

Klooz, C., 2021. What are the respective roles of state and non-state institutions in socio-technical security provision in anarchy? Analysis of today’s security provisions to identify security-relevant developments in today’s socio-technical societies, where isolated analysis of physical and cyber security is not indicative. Master’s thesis, MSc Global Corporations and Policy, Center for International Studies and Diplomacy, School of African and Oriental Studies.

Kokolakis, S., 2017. Privacy attitudes and privacy behaviour: A review of current research on the privacy paradox phenomenon. Computers & Security, 64, pp.122-134. [doi] Available at: https://doi.org/10.1016/j.cose.2015.07.002 [Accessed 31 March 2023].

Lessig, L., 2006. Code: Version 2.0. New York: Perseus Books Group. [online] Available at: https://lessig.org/product/codev2 [Accessed 30 March 2023].

Levin, S., 2019. Re: linux-distros membership application - Microsoft. [online] Available at: https://www.openwall.com/lists/oss-security/2019/06/27/7 [Accessed 31 March 2023].

Linux Foundation, 2023a. About. [online] Available at: https://www.linuxfoundation.org/about [Accessed 01 March 2023].

Linux Foundation, 2023b. Members of the Linux Foundation. [online] Available at: https://www.linuxfoundation.org/about/members [Accessed 01 March 2023].

Linux Foundation, 2023c. Leadership. [online] Available at: https://www.linuxfoundation.org/about/leadership [Accessed 01 March 2023].

Lipp, B., Blanchet, B., & Bhargavan, K., 2019. A mechanised cryptographic proof of the WireGuard virtual private network protocol. In: 2019 IEEE European Symposium on Security and Privacy (EuroS&P). pp. 231-246. IEEE. [online] Available at: https://ieeexplore.ieee.org/abstract/document/8806752 [Accessed 30 March 2023].

Margetts, H., Lehdonvirta., V., González-Bailón, S., Hutchinson, J., Bright, J., Nash., V. & Sutcliffe., D., 2021. The Internet and public policy: Future directions. Policy & Internet, 13 (2), pp.162-184. [doi] Available at: https://doi.org/10.1002/poi3.263>; [Accessed 31 March 2023]. [pre-print] Available at: https://www.researchgate.net/profile/Jonathon-Hutchinson/publication/352555279_The_Internet_and_Public_Policy_Future_Directions/links/60cfb3eea6fdcc01d48adc4e/The-Internet-and-Public-Policy-Future-Directions [Accessed 31 March 2023].

Microsoft, 2018. Microsoft Edge: Making the web better through more open source collaboration. [online] Available at: https://blogs.windows.com/windowsexperience/2018/12/06/microsoft-edge-making-the-web-better-through-more-open-source-collaboration/ [Accessed 31 March 2023].

Microsoft, 2022. Download the new Microsoft Edge based on Chromium. [online] Available at: https://support.microsoft.com/en-us/microsoft-edge/download-the-new-microsoft-edge-based-on-chromium-0f4a3dd7-55df-60f5-739f-00010dba52cf [Accessed 31 March 2023].

Milner, H., 1991. The assumption of anarchy in international relations theory: A critique. Review of International Studies, 17(1), pp.67-85. [doi] Available at: https://doi.org/10.1017/S026021050011232X [Accessed 31 March 2023].

Mitre, 2022a. CVE-2022-40768. [online] Available at: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2022-40768 [Accessed 31 March 2023].

Mitre, 2022b. CVE-2022-3435. [online] Available at: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2022-3435 [Accessed 31 March 2023].

Mozilla, 2022a. MDN Web Docs: Apple Safari. [online] Available at: https://developer.mozilla.org/en-US/docs/Glossary/Apple_Safari?retiredLocale=en [Accessed 31 March 2023].

Mozilla, 2022b. MDN Web Docs: Mozilla Firefox. [online] Available at: https://developer.mozilla.org/en-US/docs/Glossary/Mozilla_Firefox [Accessed 31 March 2023].

NIST, 2022a. CVE-2022-40768 Detail. [online] Available at: https://nvd.nist.gov/vuln/detail/CVE-2022-40768 [Accessed 31 March 2023].

NIST, 2022b. CVE-2022-3435 Detail. [online] Available at: https://nvd.nist.gov/vuln/detail/CVE-2022-3435 [Accessed 31 March 2023].

Openwall, 2022a. Openwall GNU/*/Linux. [online] Available at: https://www.openwall.com/Owl/ [Accessed 31 March 2023].

Openwall, 2022b. Openwall services. [online] Available at: https://www.openwall.com/services/ [Accessed 31 March 2023].

Openwall, 2022c. Linux kernel: information disclosure in stex_queuecommand_lck. [online] Available at: https://www.openwall.com/lists/oss-security/2022/09/09/1 [Accessed 31 March 2023].

Openwall, 2022d. Re: Linux kernel: information disclosure in stex_queuecommand_lck. [online] Available at: https://www.openwall.com/lists/oss-security/2022/09/19/1 [Accessed 31 March 2023].

Philpott, D. (1995). Sovereignty: An Introduction and Brief History. Journal of International Affairs, 48(2), pp.353–368. [JSTOR] Available at: http://www.jstor.org/stable/24357595>; [Accessed 31 March 2023].

Powell, J., 2019. Linux in Azure. [online] Available at: https://www.silversands.co.uk/blog/linux-in-azure/ [Accessed 31 March 2023].

Red Hat, 2022. Red Hat Enterprise Linux. [online] Available at: https://www.redhat.com/de/technologies/linux-platforms/enterprise-linux [Accessed 31 March 2023].

Roberts, H., Zuckerman, E., Faris, R., York, J., & Palfrey, J., 2011. The evolving landscape of Internet control. Berkman Center for Internet and Society. [pdf] Available at: https://cyber.harvard.edu/sites/cyber.law.harvard.edu/files/Evolving_Landscape_of_Internet_Control_2.pdf [Accessed 31 March 2023].

Saxenian, A., 1991. Institutions and the growth of Silicon Valley. Berkeley Planning Journal, 6(1), pp.36-57. [doi] Available at: https://doi.org/10.5070/BP36113117 [Accessed 31 March 2023].

Saxenian, A., Aoyama, Y., Powell, W., Scott, A., Storper, M., Kemeny, T., Makarem, N., and Osman, T., 2017. The Rise and Fall of Urban Economies: Lessons from San Francisco and Los Angeles. The AAG Review of Books, 5(2), pp.146-157. [doi] Available at: https://doi.org/10.1080/2325548X.2017.1292591 [Accessed 31 March 2023].

Sharwood, S., 2017. AWS adopts home-brewed KVM as new hypervisor. [online] Available at: https://www.theregister.com/2017/11/07/aws_writes_new_kvm_based_hypervisor_to_make_its_cloud_go_faster/ [Accessed 31 March 2023].

Simmons, J., 2022. WebKit Features in Safari 16.1. [online] Available at: https://webkit.org/blog/13399/webkit-features-in-safari-16-1/ [Accessed 31 March 2023].

Sowell, J., 2012. Empirical studies of bottom-up Internet governance. In: 2012 Research Conference on Communications, Information and Internet Policy. TPRC. [doi] Available at: https://dx.doi.org/10.2139/ssrn.2032285 [Accessed 31 March 2023].

Sowell, J., 2019. A conceptual model of planned adaptation (PA). In: Marchau, V. A., Walker, W. E., Bloemen, P. J., & Popper, S. W., 2019. Decision making under deep uncertainty: From theory to practice. Heidelberg: Springer Nature. pp. 289-320. https://doi.org/10.1007/978-3-030-05252-2 [pdf] Available at: https://library.oapen.org/bitstream/handle/20.500.12657/22900/1007261.pdf?sequence=1#page=293 [Accessed 31 March 2023].

Sowell, J., 2022. Mapping the Operational Institutional Complex Sustaining the Internet's Infrastructure. [online] Available at: https://www.ucl.ac.uk/steapp/research/digital-technologies-policy-laboratory/mapping-operational-institutional-complex-sustaining [Accessed 31 March 2023].

SPI, 2022. Software in the Public Interest. [online] Available at: https://www.spi-inc.org/ [Accessed 31 March 2023].

Spinics, 2018a. Re: [PATCH v2 0/5] crypto: Speck support. [online] Available at: https://www.spinics.net/lists/linux-crypto/msg32887.html [Accessed 31 March 2023].

Spinics, 2018b. Re: [PATCH v2 0/5] crypto: Speck support. [online] Available at: https://www.spinics.net/lists/linux-crypto/msg32900.html [Accessed 31 March 2023].

Spinics, 2018c. Re: [PATCH v2 0/5] crypto: Speck support. [online] Available at: https://www.spinics.net/lists/linux-crypto/msg32910.html [Accessed 31 March 2023].

Spinics, 2018d. Re: [PATCH v2 0/5] crypto: Speck support. [online] Available at: https://www.spinics.net/lists/linux-crypto/msg32911.html [Accessed 31 March 2023].

Spinics, 2018e. [PATCH v2 0/5] crypto: Speck support. [online] Available at: https://www.spinics.net/lists/linux-crypto/msg33291.html [Accessed 31 March 2023].

Statcounter, 2023. Mobile & Tablet Operating System Market Share Worldwide. [online] Dublin: StatCounter. Available at: https://gs.statcounter.com/os-market-share/mobile-tablet/worldwide/#monthly-202004-202004-bar [Accessed 30 March 2023].

Statista, 2022. Amazon, Microsoft & Google Dominate Cloud Market. [online] Available at: https://www.statista.com/chart/18819/worldwide-market-share-of-leading-cloud-infrastructure-service-providers/ [Accessed 03 March 2023].

Vaughan-Nichols, S., 2018. Linux now dominates Azure. [online] Available at: https://www.zdnet.com/article/linux-now-dominates-azure/ [Accessed 31 March 2023].

W3Techs, 2021a. Usage statistics of Unix for websites. [online] Maria Enzersdorf: Q-Success. Available at: https://w3techs.com/technologies/details/os-unix [Accessed 30 March 2023].

W3Techs, 2021b. Usage statistics of operating systems for websites. [online] Maria Enzersdorf: Q-Success. Available at: https://w3techs.com/technologies/overview/operating_system [Accessed 30 March 2023].

Wheeler, D., 2021. Preventing Supply Chain Attacks like SolarWinds. [online] Available at: https://www.linuxfoundation.org/en/blog/preventing-supply-chain-attacks-like-solarwinds/ [Accessed 18 February 2023].