CyBOK

DISTRIBUTED SYSTEMS SECURITY KNOWLEDGE AREA

Issue 1.0

AUTHOR: Neeraj Suri – Lancaster University

EDITOR: Emil Lupu – Imperial College London

REVIEWERS:

Konstantin Beznosov – University of British Columbia

Marko Vukolić – IBM Research

The CyBOK project would like to understand how the CyBOK is being used and its uptake. The project would like organisations using, or intending to use, CyBOK for the purposes of education, training, course development, professional development etc. to contact it at contact@cybok.org to let the project know how they are using CyBOK.

Issue 1.0 is a stable public release of the Distributed Systems Security Knowledge Area. However, it should be noted that a fully collated CyBOK document which includes all the Knowledge Areas is anticipated to be released in October 2019. This will likely include updated page layout and formatting of the individual Knowledge Areas.

Distributed Systems Security

Neeraj Suri

October 2019

INTRODUCTION

A distributed system is typically a composition of geo-dispersed resources (computing and communication) that collectively (a) provides services that link dispersed data producers and consumers, (b) provides on-demand, highly reliable, highly available, and consistent resource access, often using replication schemas to handle resource failures, and (c) enables a collective aggregated capability (computational or services) from the distributed resources to provide (an illusion of) a logically centralised/coordinated resource or service.

Expanding on the above, the distributed resources are typically dispersed (for example, in an Azure or Amazon Cloud, in Peer-to-Peer Systems such as Gnutella or BitTorrent, or in a Blockchain implementation such as Bitcoin or Ethereum) to provide various features to the users. These include geo-proximate and low-latency access to computing elements, high-bandwidth and high-performance resource access, and especially highly-available uninterrupted services in the case of resource failure or deliberate breaches. The overall technical needs in a distributed system consequently relate to the orchestration of the distributed resources such that the user can transparently access the enhanced services arising from the distribution of resources without having to deal with the technical mechanisms providing the varied forms of distributed resource and service orchestrations.

To support these functionalities, a distributed system commonly entails a progression of four elements. These include (a) data flows across the collection of authorised inputs (regulated via Access/Admission Control), (b) transportation of the data to/across the distributed resources (Data Transport functionality), (c) a resource coordination schema (Coordination Services), and (d) property based (e.g., time or event based ordering, consensus, virtualisation) data management to support the desired applications such as transactions, databases, storage, control, and computing.

Consequently, distributed systems security addresses the threats arising from the exploitation of vulnerabilities in the attack surfaces created across the resource structure and functionalities of the distributed system. This covers the risks to the data flows that can compromise the integrity of the distributed system's resources/structure, access control mechanisms (for resource and data accesses), the data transport mechanisms, the middleware resource coordination services characterising the distributed system model (replication, failure handling, transactional processing, and data consistency), and finally the distributed applications based on them (e.g., web services, storage, databases and ledgers).

This Knowledge Area first introduces the different classes of distributed systems categorising them into two broad categories of decentralised distributed systems (without central coordination) and the coordinated resource/services type of distributed systems. Subsequently, each of these distributed system categories is expounded for the conceptual mechanisms providing their characteristic functionalities prior to discussing the security issues pertinent to these systems. As security breaches in a distributed system typically arise from breaches in the elements related to distribution (dispersion, access, communication, coordination, etc.), the KA emphasises the conceptual underpinnings of how distributed systems function. The better one understands how functionality is distributed, the better

one can understand how systems can be compromised and how to mitigate the breaches. The KA also discusses some technology aspects as appropriate along with providing references for following up the topics in greater depth.

CONTENT

1 Classes of Distributed Systems and Vulnerabilities

[1, c2][2, c5][3, c18]

1.1 Classes of Distributed Systems

A diversity of viewpoints, models, and deployments exist for characterising distributed systems. These include defining a distributed system at the level of the aggregation of physical resources (e.g., Peer to Peer or Cloud systems), defining it at the Middleware level (e.g., Publish-Subscribe, distributed object platforms, or Web services), or defining it in terms of the services a distributed system provides (e.g., Databases or Ledgers). While a spectrum of definitions exists in literature, distributed systems can be broadly classified by the coordination schema linking the resources or by the specification of the services utilising them. One broad class is of *decentralised control* where the individual resources primarily interact with their "neighbouring" resources. The other broad category links the distributed resources via communication processes, such as message passing, to realise varied forms of *virtual centralised/coordinated control*. Thus, based on such communication and coordination models, distributed systems can be categorised into the following two broad classes.

1. *Decentralised point-to-point interactions across distributed entities without a centralised coordination service*: Peer to Peer (P2P) systems represent this class of distributed systems. Decentralised un-timed control is a prominent characteristic of such systems. For example, systems such as Kademlia, Napster, Gnutella, and many other distributed file and music sharing/storage systems, wireless sensor networks as well as online gaming systems fall in this category.

2. *Coordinated clustering across distributed resources and services*: This is a broad class that is best understood when sub-divided into two coordination sub-classes, namely (a) the coordination of resources and (b) the coordination of services. We will utilise these two coordination abstractions throughout this chapter. The spectrum of distributed systems includes Client-Server models, n-Tier Multi-tenancy Models, elastic on-demand geo-dispersed aggregation of resources (Clouds – public, private, hybrid, multi-Cloud, Big Data services, High Performance Computing), and transactional services such as Databases, Ledgers, Storage Systems, or Key Value Stores (KVS). The Google File System, Amazon Web Services, Azure, and Apache Cassandra are simple examples of this class. While this class may appear to be both broad and diverse, the coordination abstraction (for either resources or services) directly characterises the type of distributed system into these two sub-classes. In both cases, these systems are typically coordinated via communication exchanges and coordination services with the intended outcome of providing a "virtually centralised system" where properties such as causality, ordering of tasks, replication handling, and consistency are ensured. There are discrete definitions in the literature for Client-Server systems, Cloud Computing, Mobile Computing, Distributed Databases, etc., though the provisioning of virtual "centralised/coordinated" behaviour is a common characteristic across them.

Notes:

There are many nuances of security in distributed systems. One viewpoint focuses on the concepts and mechanisms to provide security in a distributed system where the resources and services are

dispersed. The other viewpoint considers using distribution as a means of providing security, e.g., the dispersal of keys versus a centralised key store or the use of Virtual Machines (VMs) to partition and isolate resources and applications. This KA focuses on the former category of "security in a distributed system". However, it also discusses the latter viewpoints given that the dispersed security mechanisms typically execute on dispersed resources logically resulting in the need for the above mentioned classes of Decentralised or Coordinated clustering.

It is worth highlighting that a distributed system architecture is often an aggregation of multiple layers where each layer builds upon the services provided by the layer below and coordinated services offered across the distribution. At the lowest level, resources within a particular device (memory, computation, storage, communication) are accessed through the Operating System primitives provided on that device. Distributed services e.g., naming, time synchronisation, distributed file systems are assembled through the interaction of different components and services running on individual devices. Higher layers build upon the lower layers and services to provide additional functionalities and applications. Interactions across the different components of the distributed system at each level are provided by middleware frameworks that support many different communication styles: message passing, Remote Procedure Calls (RPCs), distributed object platforms, publish-subscribe architectures, enterprise service bus. Distributed applications are thus realised in a layered (or tiered) fashion through the interactions and coordination of distributed components and services. Within these architectures, decentralisation and coordination at each layer may differ resulting in hybrid compositions of decentralisation and coordination patterns. We refer the reader to the Operating Systems and Virtualisation KA for issues concerning access to basic resources and the books [4, 5, 3, 6, 7] for further reading on distributed systems architectures and middleware.

1.2 Classes of Vulnerabilities & Threats

Vulnerabilities refer to design or operational weaknesses that allow a system to be potentially compromised by an attacker. Analogously, a threat reflects the potential or likelihood of an attacker causing damage or compromising the system. Furthermore, security is an end-to-end systems property. Consequently, the vulnerabilities of a distributed system are broadly grouped based on the functional blocks therein defining the distributed system. Logically, these functional blocks and their operations also constitute the threat/attack surface for the systems where an attacker/adversary can exploit a vulnerability to compromise the system. At a high level, the attack surface relates to the compromises of the physical resources, the communication schema, the coordination mechanisms, the provided services themselves, and the usage policies on the data underlying the services.

The following outlines the general functionalities that will be progressively detailed in the subsequent sections as relevant to the specific distributed system model.

1.2.1 Access/Admission Control & ID Management

Access or Admission control determines the authorised participation of a resource, a user, or a service within a distributed system. This can include the sourcing of data and the access rights to read/write and use data over the lifetime of a service. The potential threats and consequent attacks include masquerading or spoofing of identity to gain access rights to the data. They can also involve Denial of Service (DoS) attacks that detrimentally limit access (e.g., depletion of computing resources and communication channels) leading to the inaccessibility and unavailability of the distributed resources/services. It is worth emphasising that resource distribution often entails more points for access control, and also more information transported in the system to support access control thus increasing the attack surface of the system (see the AAA KA for a discussion of authentication and authorisation in distributed systems).

A distributed system entity (resource, service, user, or data element) participates in a distributed system with a physical or logical identity. The identity, statically or dynamically allocated, can be a

resource identifier such as an ID name or a number[1]. Here, authorisation may be specified in terms of the user and/or resource identity including the use of login names and passwords. Thus, an activity that involves tampering with the identity constitutes a likely threat.

1.2.2 Data Transportation

The network level threats span routing, message passing, the publish-subscribe modalities of resource interaction, event based response triggering, and threats across the middleware stack. Moreover, these can be passive (eavesdropping) or active attacks (data modification). A typical example is the Man In the Middle (MITM) attack where the attacker inserts itself between the victim's browser and the web server to establish two separate connections between them. This enables the attacker to actively record all messages and selectively modify data without triggering a suspicious activity alarm if the system does not enforce endpoint authentication. We refer the reader to [4, 5] for detailed coverage of these topics, and to the Network Security KA.

1.2.3 Resource Management and Coordination Services

This critical group encompasses the spectrum of threats to the mechanisms (typically middleware protocols) that provide the coordination of resources. This includes, among others, the aspects of synchronisation, replication management, view changes, time/event ordering, linearisability, consensus, and transactional commit.

1.2.4 Data Security

As a distributed system essentially operates on data (at rest or in motion) over the facets of data-sourcing, data-distribution, data-storage, or data-usage in services, the classical CIA (Confidentiality, Integrity and Availability) properties directly apply to each element (and interfaces) of this data chain. The threats to confidentiality include information leakage threats such as Side Channel Attacks or Covert Channel Attacks. Any delay or denial of data access constitutes a threat to Availability. Integrity aspects concern any compromise of data correctness such as the violation of data consistency as observed by the distributed participants. This includes the different types of consistency (strong, weak, relaxed, eventual, etc.) over storage and transactional services. Consequently, addressing the security of the data elements of a distributed system requires consideration of the threats mentioned above across resources, access control, data transportation, and coordination services as well as data threats in the form of malicious applications, code, and viruses (see Malware KA).

Section Organisation

Based on this overview, the subsequent sections progressively outline the security approaches for distributed systems as split into the above mentioned classes of decentralised and coordination based systems. In order to understand the security issues relevant to each class, the sections also provide a basic overview of the underlying distributed system concepts along with pointers for further reading. Section 2 presents the commonly used models for decentralised P2P systems. Section 3 then elaborates the corresponding security threats for the P2P systems. This is followed by the exposition of coordinated distributed system models in Section 4, and by a discussion of the corresponding security aspects in Section 5.

2 Distributed Systems: Decentralised P2P Models

[8, c11-12][2, c25]

Peer-to-Peer (P2P) systems constitute a decentralised variant of distributed systems. Their popularity is driven by the characteristic P2P features of scalability, decentralised coordination, and low

[1] [6] provides an excellent discourse on naming issues in Chapter 6.

cost. Scalability implies that no changes to the protocol design are needed with increasing numbers of peers. Whereas a Client-Server architecture typically entails increasing back-end (Server) resources with increasing numbers of (Client) requests, this is not the case in P2P systems due to their inherent decentralised architecture. Furthermore, the decentralised P2P system designs promote inherent resilience against individual peer failures or other disruptions. The peer population itself represents the service provisioning infrastructure of the system. Thereby, potential service consumers are required to partake in resource provisioning avoiding the need for dedicated data centres. Over the past two decades, a multitude of P2P models have emerged. Regardless of their specific realisation, they usually combine the following five principles: (1) symmetry of interfaces as peers can take inter-changeable duties as both servers and clients, (2) resilience to perturbations in the underlying communication network substrate and to peer failures, (3) data and service survivability through replication schemes, (4) usage of peer resources at the network's edge, imposing potentially low infrastructure costs and fostering scalability as well as decentralisation, and (5) address variance of resource provisioning among peers.

These five principles make P2P a vital foundation for a diverse set of applications. Originally, P2P systems were (in)famous for their support of file sharing applications such as eMule or KaZaA, though their usage is now common in applications such as social networks, multimedia content distribution, online games, internet telephony services, instant messaging, the Internet of Things, Car-to-Car communication, supervisory control and data acquisition (SCADA) systems, and wide area monitoring systems. As discussed in later sections, distributed ledgers also utilise some aspects of P2P operations.

P2P Protocol Categories

The two major P2P paradigms are *unstructured* and *structured* systems. These system designs directly correlate with the application categories introduced in the previous section, i.e., unstructured protocols are mostly suitable for (large scale and scalable) data dissemination, whereas structured ones are usually applied for efficiency of data discovery. The emergent hybrid P2P protocol designs combine aspects from both unstructured and structured ones within an integrated P2P system.

Additionally, hierarchical P2P systems also exist. These partly contradict the conceptual P2P principle that considers all peers as *equal* in the sense of service provisioning. These hierarchical systems can be considered as layered systems, e.g., composition of multiple overlays consisting of front-end and back-end peers.

Regardless of the type of P2P system, it is important to note that the basic P2P operations are based on three elements, namely (a) identification or naming of peer nodes, (b) routing schemas across peers, and (c) discovery of peers as a function of their identifiers and routing.

In order to support the discussion of security in P2P systems, the next subsections provide an introductory level technical overview on P2P protocols. We provide a brief overview of the P2P protocol categories in regard of the overlay topology, resources discovery, and message passing. The reader is referred to [9] for a comprehensive discussion on P2P operations.

2.1 Unstructured P2P Protocols

Representatives of the unstructured P2P protocol class such as Freenet[2] or Gnutella [10, 11] are mainly used for data dissemination applications such as censorship-free[3] communication or file sharing. While the set of peers do not have any characteristic topology linking them, their implicit topology is usually embedded within the physical communication underlay network topology and often unveils

[2] https://freenetproject.org/

[3] In the sense that data and information is stored and exchanged with integrity and privacy preserving techniques to address freedom of expression and speech concerns.

tree or mesh like sub-graphs, which allow for low latency message exchange, e.g., to address timeliness requirements of data dissemination applications. Tree topologies can be found, e.g., in single source streaming media data dissemination with various consumers as leaf nodes. Meshes are the more generic case, for example, in applications with multiple sources and sinks such as in file sharing applications.

Unstructured P2P protocols typically search for resources (i.e., peers and data) by name or labels, and do not use a structured addressing scheme. This feature supports scalable dissemination but scales poorly for resource discovery or reproducible routing paths. Peers nevertheless maintain an identifier to allow independence of the underlay network address. Resources are discovered using search algorithms on the overlay graph. Examples of search algorithms include breadth-first search, depth-first search, random walks, or expanding ring searches. These options are often combined according to the requirements of the application.

The communication across peers is via messages. Message passing may be direct, i.e., using an underlay network connection between two peers, but this usually requires that the peers explicitly know the peer address and route. When the destination peer for the message to be sent is unknown, messages are piggybacked alongside a resource discovery operation.

All peers maintain lists (direct routing tables with addresses or hashed addresses) with contact information about other peers. Hence, messaging works efficiently and the network does not suffocate from address-search messages. The efficiency of such lists depends on the liveness of the peers. Hence, the listed peers are periodically pinged for liveness and removed when no reply is received. The periodicity is dynamically adjusted based on the relevant churn, i.e., the rate of peer joins and departures.

2.2 Structured P2P Protocols

Structured P2P protocols such as Chord, Pastry, Tapestry, Kademlia, CAN etc. [12, 13, 14, 15] are typically used for data discovery applications where the structure of the topology aids efficient searches. Their topology graphs usually show small-world properties, i.e., there exists a path between any two peers with a relatively small number of edges. Structured topologies often appear as ring structures with shortcuts, which forms a basis for scalable and efficient operations such as resource discovery and message passing. Some protocols have more exotic topologies, e.g., butterfly graphs, fixed-degree graphs, or a multi-torus. The salient characteristics are efficiency of node discovery and efficiency of routing that uses information on the P2P structure and topology. As this aspect has security implications, we briefly detail these operations.

Unlike unstructured P2P's open addressing schemas, in structured P2P protocols, pointers to resources such as peers or data are stored in a distributed data structure which is called a *distributed hash table (DHT)*. The overlay's *address space* is usually an integer scale in the range of $[0, \ldots, 2^w - 1]$ with w being 128 or 160 in general. Usually, a *distance function* $d(a, b)$ is defined which allows distance computations between any two identifiers a and b in the address space. Distance computations are crucial for the lookup mechanism and data storage responsibilities. The distance function and its properties differ among protocol implementations. Data discovery is realised by computing the key of an easy-to-grasp resource identifier such as a distinctive name/key and subsequently requesting that key and its data from one of the responsible peers.

Messages – for example to request the data for a given key – are exchanged in most structured protocols directly, i.e., using an underlay network connection between two peers. If peers do not know each other, then no direct connection can be set up and the destination peer's location needs to be determined to conduct routing. To this end, an overlay lookup mechanism aims to steadily decrease the address space distance towards the destination on each iteration of the lookup algorithm until the identifier can be resolved. This design approach turns out to be very efficient and promotes scalability. Once the lookup has successfully retrieved the destination's underlay network address,

messages can be exchanged. Lookup variants include iterative or recursive algorithms as well as parallelised queries to a set of closest neighbour peers.

Routing tables usually store $k \cdot w$ entries with k being a protocol specific constant. Moreover, for the i^{th} portion of k entries with $i \in [0 \dots w]$, the peer stores contact information of peers that share i common prefix bits of the peer's key. In other words, routing tables usually provide more storage for closer peers than more distant ones. Moreover, routing tables keep only information about live and reachable peers, therefore peers are periodically pinged. In structured protocols, maintenance is more expensive as the topological structure needs to be retained, e.g., newly joined peers have to be put into the appropriate peer's routing tables or leaving/unresponsive peers have to be replaced by live ones in many peers' routing tables.

2.3 Hybrid P2P Protocols

Hybrid variants of P2P protocols integrate elements from unstructured and structured schemas, as their principal intent is data discovery and data dissemination. Prominent hybrid protocol examples include file sharing services such as Napster and BitTorrent [16]. BitTorrent was originally a classical unstructured protocol but now has been extended with structured P2P features to provide a fully decentralised data discovery mechanism. Consequently, BitTorrent could abandon the concept of so called "tracker servers" (that facilitated peer discovery) and improve its availability. On the other hand, architectural requirements often need to be considered to fully utilise the capacity of hybrid P2P protocols. An example would be establishing how the data discovery is transmitted among the servers and how it is reported back to the user [17]. Similar considerations apply to other streaming overlay approaches.

2.4 Hierarchical P2P Protocols

Typically, all the peers in a P2P system are considered to be *equal* in terms of the client-server services they can provide. Yet, for some application scenarios it turns out that a hierarchical P2P design can be advantageous. These can include a layered design of structured and unstructured overlays. In hierarchical designs, peers are further categorised based on their bandwidth, latency, storage, or computation cycles provisioning with some (super) peers taking a coordinating role. Usually, the category with fewer peers represented the back-end part of the hierarchical system, whereas the multitude of peers act as front-end peers that process service requests at the first level and only forward requests to the back-end when they cannot fulfill the service request in the first place. This improves the look-up performance and also generates fewer messages in the network. Furthermore, popular content can be cached locally to reduce download delays [18]. This design has proven successful, for example, in the eDonkey file sharing system or in Super P2P models such as KaZaA where a selected peer acts as a server to a subset of clients.

3 Distributed Systems: Attacking P2P Systems

[3, c16][19, c5]

We present security attacks corresponding to the above mentioned classes of P2P systems. To facilitate this discussion, we outline the functional elements of a P2P system that help the reader relate the security implications for specific systems or application cases. Subsequently, we assess the risks stemming from attacks to plan the requisite mitigation. The P2P functional elements that need protection broadly include:

1. *P2P Operations* (P-OP) such as discovery, query, routing, download, etc. that are accessible through the service interface of the P2P protocol. This functionality relates to the network level.

2. *P2P Data Structures* (P-DS), e.g., data stored in a peer's routing table or resources that are shared with other peers of the overlay network. This functional element may be accessible at either the network level or locally on the peer's host machine.

We will refer to these two elements as P-OP and P-DS, in the following subsections where we discuss the specific P2P attacks. We use the established security notions of [20] for Confidentiality, Integrity and Availability. Whenever a definition refers to authentication, we assume that peers are implicitly authenticated on joining the overlay network. P2P protocols may may use admission control systems or may be open to arbitrary peers.

Note that we focus on attacks *against* P2P systems (e.g., denial of service or routing disruptions) and do not consider attacks that are prepared or conducted *using* P2P systems in order to harm non-P2P systems (e.g., using a P2P system to coordinate distributed denial of service attacks).

3.1 Attack Types

We now present the different attacks that are specific to P2P systems. Broadly, the attacks correspond to attacking the functional elements, P-OP and P-DS, either by (a) disrupting their connectivity or access to other nodes for dissemination/discovery/routing or (b) corrupting their data structures. Besides the well known (distributed) denial of service attacks which apply to P2P as well as to other systems, most attacks exploit fundamental P2P features such as message exchange based decentralised coordination and especially that each peer has only a partial (local) view of the entire system. Consequently, attackers aim to trick other peers by providing incorrect data or collude to create partitions that hide views of the system from good nodes. This includes example scenarios such as (a) to mislead peers in terms of routing, (b) to take advantage of access to resources, (c) to overcome limitations in voting systems or games, or (d) to hide information in the overlay among others. We refer the reader to the survey articles [21, 22] for a fuller exposition of P2P security. We now enumerate some representative security attacks and relate them to their corresponding impact on Confidentiality, Integrity, and Availability (CIA). Some examples of attacks are further discussed in Section 3.2 along with corresponding mitigation approaches.

- *Denial of service attacks* (DoS) [20], *distributed denial of service attacks* (DDoS), or *disruption attacks* [23] manifest as resource exhaustion by limiting access to a node or a communication route. In P2P architectures, the attacker aims to decrease the overlay network's service availability by excessively sending messages to a specific set of peers and thereby negatively affecting the P-OP functionality. This could affect the peer join/leave mechanism, or other arbitrary P2P service aspects, e.g., damaging the routing put/get operations in a DHT. For example, benign peers may be impaired by an excessive maintenance workload. Moreover, DoS and DDoS attacks can have a negative impact on bandwidth usage and resource provisioning which may result in degraded services. For instance, GitHub was hit with a sudden onslaught of traffic that reached 1.35 terabits per second[4]. The traffic was traced back to "over a thousand different autonomous systems (ASNs) across tens of thousands of unique endpoints" participating in the attack.

- *Collusion attacks* [24] aim to compromise the availability, integrity, or confidentiality of P2P networks. Collusion refers to the fact that a sufficiently large subset of peers colludes to carry out a strategy which targets the P2P services and thereby negatively affects the P-OP functionality. The typical attack aims to override control mechanisms such as those for reputation or trust management, or bandwidth provisioning. The Sybil and Eclipse attacks, discussed later on, are based on attackers colluding to create network partitions to hide system state information from good nodes.

- *Pollution attacks* [25, 26] or *index poisoning* [27] aim to compromise the P2P system's integrity and its P-DS functionality by adding incorrect information. Consequences of pollution attacks are the proliferation of polluted content resulting in service impairments. An example is the typhoid ad-ware

[4] https://www.wired.com/story/github-ddos-memcached

attack where the attacker partially alters the content, e.g., adding advertisement at a single peer that subsequently spreads this polluted content to other peers.

- *White washing* [26] or *censorship attacks* aim to compromise the availability or integrity of P2P systems. This includes either illicit changing of, deletion of or denying access to data. Therefore, these attacks endanger the P-DS functionality. White washing attacks are especially dangerous for P2P systems that use reputation based systems since they allow a peer with a bad reputation to leave the system, and subsequently re-join as a benign user.

- *Routing attacks* [28, 23] aim to compromise the availability or integrity of P2P networks. Routing attacks play an important role in composite attacks, such as the Eclipse attack which obstructs a good node's view of the rest of the system. In routing attacks, a malicious peer undermines the message passing mechanism, e.g., by dropping or delaying messages. Another routing attack variant is *routing table poisoning (RTP)* [28]. In this attack, an attacker deliberately modifies its own or other peers' routing tables, e.g., by returning bogus information to benign peer lookup requests. *Attraction and repulsion* [23] are specific variants of routing attacks which either increase (attraction) or decrease (repulsion) the attractiveness of peers, e.g., during path selection or routing table maintenance tasks. These attacks negatively affect the P-DS functionality. The compromise of the routing table in Pastry, often used in online social networks, is a typical routing attack.

- *Buffer map cheating attacks* [29] aim to decrease the availability of P2P networks, particularly those used for media streaming applications. Through this attack, adversaries reduce the outgoing traffic load of their peers by lying about their data provisioning. This is also an infringement on integrity and affects the P-OP functionality. This attack is especially relevant in streaming media P2P applications which rely on the collaboration of peers. Omission, Fake Reporting, Fake Blocks, incorrect Neighbour Selection are related implications of such attacks.

- *Sybil attacks* [30] aim to compromise the availability or confidentiality (via spoofing) of P2P networks and can be regarded as a specific version of *node/peer insertion attacks.* They consider the insertion into the overlay of peers that are controlled by one or several adversaries. This could happen at specific or arbitrary locations of the overlay's topology, depending on the attacker's aim. Furthermore, P2P applications may consider system users as legal entities and consequently restrict the amount of peers per user to the amount of allowed votes for that entity. Hence, an imbalance results in terms of the expected amount of peers per user. Sybil attacks may be a precursor for many of the previously described attacks. Sybil attacks affect the P-OP functionality of the system. Prominent Sybil attacks include the compromise of the BitTorrent DHT and the Sybil attack on the Tor anonymisation network.

- *Eclipse attacks* [31] aim to decrease the availability, integrity and confidentiality of P2P networks. Essentially, a good peer is surrounded by a colluding group of malicious peers that either partially or fully block the peer's view of the rest of the system. The consequence is that the malicious nodes can either mask or spoof the node's external interactions. This is a composite attack that may involve routing table poisoning, DoS/DDoS, Sybil attacks, collusion, white washing, or censorship. Consequently, these attacks have an impact on both the P-OP and P-DS functionality. Variants of Eclipse attacks include Localised Eclipse Attacks (LEA), Topology Aware Localised Eclipse Attacks (taLEA) and Outgoing Eclipse Attacks (OEA) attacks among others. An example of an Eclipse attack on Bitcoin is discussed in Section 5.

3.1.1 Summary

Table 1 summarises attacks on the P2P functional elements that entail modifications of the P2P system to either degrade or compromise the P2P operations. The adversarial collusion of malicious peers is a key factor to launch these attacks resulting in significant disruption. In many cases, the inherent design choices of P2P, which foster scalability and fault tolerance, are exploited. Attacks against P2P systems usually show an impact in terms of the system's confidentiality, integrity, or availability. Several of the observed attacks are known from other system architectures such as

Attack	Availability	Integrity	Confidentiality	Functionality
DoS/DDoS	✓	✗	✗	P-OP
Collusion	✓	✓	✓	P-OP
Pollution	✗	✓	✗	P-DS
White washing & censorship	✓	✓	✗	P-DS
Routing	✓	✓	✗	P-DS
Buffer map cheating	✓	✓	✗	P-OP
Sybil	✓	✗	✓	P-OP
Eclipse	✓	✓	✓	P-DS, P-OP

Table 1: P2P Attacks, Security Goals and Affected Functionality

client-server models while others are new ones or compositions of various attacks. The difference from comparable attacks in client-server system architectures is that P2P overlay networks may grow very large and adversaries have to correspondingly adapt their efforts, i.e., they need to scale up the fraction of malicious peers accordingly, thereby requiring a substantial amount of coordination to execute an effective collusion strategy. These attacks vary depending upon whether the attacker has direct or indirect network access via a P2P overlay. The latter requires attackers to properly join the network prior to the attack. Thus, this may entail malicious peers making, e.g., a proper announcement in the overlay network, before they can launch their adversarial behaviour.

Supplemental Observations:

- Denial of service attacks degrade or prevent a system from correct service delivery [32, 33]. The more sophisticated Sybil attack [34, 33, 35] can be used as a potential precursor for an Eclipse attack [34, 33].

- If either secure storage, secure routing, or authentication mechanisms cannot be provided, a set of attacks including omission, content forgery, content pollution, censorship, or routing table poisoning may be the consequence [33, 35].

- Churn relates to the effects of peers joining and leaving in an overlay. Churn attacks consider artificially induced churn with potentially high peer join/leave rates to cause bandwidth consumption due to the effort needed to maintain the overlay structure. This can lead to partial or complete denial of service [35].

- Varied cheating attack strategies exist (for observing or corrupting player information and activities) in massive multiplayer online games (MMOG) built upon P2P architectures [35].

3.2 Attacks and their Mitigation

We present some example attacks along with the approaches used to mitigate them. For a comprehensive coverage, we refer the reader to the surveys of [21, 22].

Basic P-OS and P-DS Based Scenarios: The prominent P2P protocol security mechanisms are authentication mechanisms, secure storage, and secure routing. These three mechanisms allow the implementation of various downstream mechanisms. Authentication mechanisms [36, 33] help to maintain a benign peer population and provide the technical basis for downstream mechanisms like secure admission, secure storage, or secure routing. Secure storage is vital for data centric applications in order to prevent attackers from conducting illicit data modifications [32, 34, 37, 36]. In a broader sense, illicit data modification in online games is considered as cheating [35]. The use of secure routing is typically advocated as an approach to facilitate the identification of peers conducting improper message forwarding [34, 37, 36]. Limiting the number of routing paths and/or protecting the paths using (high overhead) cryptographic approaches are alternate approaches to mitigating routing attacks.

Sybil and Eclipse Scenarios: Sybil attacks occur where the attacker could launch an attack with a small set of malicious peers and subsequently gather multiple addresses, which allows malicious peers to fake being a larger set of peers. Using Sybil attacks, a LEA can be launched via a chain of Sybil/malicious nodes. However, the attack relies on the assumption of the existence of a single path towards the victim that can be manipulated by the attacker. Alternately, a LEA can be launched using Sybil peers.

In such attacks, mitigation relies on using a centralised authority that handles peer enrolments or admission. Extending this concept, adding certificates (issued by a common Certificate Authority) to peers' network IDs while joining the network is another possibility. Other mitigation techniques to prevent malicious entities from selecting their own network IDs could entail a signing entity using public key cryptography.

Buffer Map Cheating Scenarios: Other disruptions could be used to attack the KAD P2P network [15], which is a Kademlia based network, through flooding peer index tables close to the victim with false information as a simplistic taLEA variant. A KAD network crawler is introduced to monitor the network status and detect malicious peers during a LEA. However, a high overhead is incurred if each peer uses such a mechanism to detect malicious entities. This becomes impractical as the overlay size increases.

Divergent lookups have been proposed as an alternate taLEA mitigation technique where the disjoint path lookups avoid searching the destination peer's proximity to skip the wasteful querying of malicious peers under taLEA assumptions.

Routing Scenarios: Mitigation mechanisms to handle routing attacks consider assigning multiple paths for each lookup using disjoint paths though at the cost of high message overhead. Alternatives include the use of cryptographic schemes to protect the paths. However, P2P is a decentralised coordination environment where implementing a centralised service to support the coordination of system wide cryptographic signatures is hard to realise.

The aforementioned security mechanisms increase the resilience of P2P systems against the various attacks. Naturally, these mechanisms are resilient only until a critical mass of colluding malicious peers is reached. In addition, some of these mechanisms require cryptographic support or the identification of peers. These requirements may interfere with application requirements such as anonymity, heterogeneity, or resource frugality.

4 Distributed Systems: Coordinated Resource Clustering

[8, c5,7,12,25][1, 3][2, c5,c14] [3, c16-17,c19]

Contrasting with the decentralised-control of P2P systems, a multitude of distributed systems exist where the interactions across the distributed resources and services are orchestrated using varied coordination mechanisms that provide the illusion of a logically centralised and coordinated system or service. The coordination can simply be a scheduler/resource manager, a discrete coordinator or a coordination group, and include ordering in time (causality) or varied precedence orders across distributed transactions. While it is tempting to define each type of distributed system discretely (i.e., differing from decentralised control in P2P), the large and diverse group of distributed systems/services share a common abstraction of "coordination" although its realisation and resultant properties for each system will vary.

Firstly, there is the case where a service is replicated on a distributed resources platform (or infrastructure) to enable geo-dispersed access to users while sustaining the required type of consistency specifications on the service. The Cloud and many distributed Client-Server systems fall in this category.

The alternate approach addresses distributed services (versus platforms) where the dispersed service participants interact to yield the collective distributed service for given consistency requirements.

For example, transactional databases and distributed ledgers fall in such a category of strong consistency. Web crawlers, searches, or logistics applications may well work with weak consistency specifications.

Overall, these constitute the two broad classes of distributed systems in the coordinated resource pooling mode, namely the classes of *resource-coordination* and *service-coordination*, as based on their characteristic coordination schema although their functionality and definitions often overlap.

In the subsequent subsections, in order to contextualise distributed systems security, we first detail the basic distributed concepts along with the coordination schema based on them. This is followed by outlining the characteristic systems in each of the resource and service coordination models. This forms the basis behind the general set of disruptions/vulnerabilities relevant to both classes of coordinated distributed systems. We then outline the threats and security implications specific to each class of systems. We refer the reader to the excellent texts of [2, 8, 1] for a comprehensive and rigorous treatise of these issues.

A Note on Technologies Underlying Distributed Platforms:

The introduction emphasised that the focus of this KA is on security in distributed systems rather than the use of distribution towards providing security. Expanding on this topic, it is worth commenting on alternate perspectives related to the "design and realisation" of distributed platforms and services. This design oriented perspective tends to emphasise the architecture of distributed systems, distributed services and their construction. This perspective typically focuses on (a) establishing security requirements, (b) realisation approaches on how to meet given security requirements at each level of abstraction, and (c) considers a distributed system as a layered architecture where each layer builds upon the primitives offered at the layer below and from distributed services. In this perspective, centralised (coordinated) and decentralised patterns are often combined, differently and at different layers. Also from this perspective, the security requirements of the applications must be met by complementing and building upon what is offered at the lower layers and services.

This is a construction and compositional approach where the security properties (requirements) at the application level, or at a given layer, drive the selection of solutions and subsystems that must be assembled (e.g., authentication, authorisation, accountability, non-repudiation etc.). The composition of such subsystems/solutions is often achieved through the use of trade-offs (and also threat) analysis that tend to cover some and not all of the requirements and thus determining relative strengths and weaknesses. For example, blockchain applications, further discussed in Section 5.2, emphasise non-repudiation and decentralisation as their main properties.

This layered and compositional approach can often be encountered in the literature such as [7, 38, 4, 5, 3, 2, 39] and many others. As the architectures and realisation fundamentally underlie the KA premise of providing security in distributed systems, the reader is encouraged to refer to this literature. The following section returns the focus back on distributed system concepts, and especially the fundamental concepts of the coordination class of distributed systems.

Distributed Concepts, Classes of Coordination

As mentioned in the introduction, a distributed system is a collation of geo-dispersed computing resources that collectively interact to provide (a) services linking dispersed data producers and consumers, (b) high-availability via fault tolerant replication to cover resource (computing and communication) failures, or (c) a collective aggregated capability (computational or services) from the distributed resources to provide (an illusion of) a logically centralised/coordinated resource or service.

Distributed systems are often structured in terms of services to be delivered to clients. Each service comprises and executes on one or more servers and exports operations that the clients invoke by making requests. Although using a single, centralised server appears tempting, the resulting service

resident on a server can only be as fault tolerant as the server hosting it. Typically, in order to accommodate server failures, the servers are replicated, either physically or logically, to ensure some degree of independence across server failures with such isolation. Subsequently, replica management protocols are used to coordinate client interactions across these server replicas. Naturally, the handling of client failures or client compromises (including their role in launching attacks via malicious code or viruses) also needs to be considered.

We now outline a basic set of distributed system concepts that also constitute the basis of the security considerations therein. The concepts are presented at an informal level to communicate the intuitions, and the reader is referred to [8, 1, 2, 3] for a comprehensive treatise on the topics.

4.1 Systems Coordination Styles

In order for the distributed resources and services to meaningfully interact, the synchronisation basis across them, in physical time or in logical order, needs to be specified. The synchronisation applies at both the network and process levels. We refer the reader to [3, 2, 1, 8] for more details. At a high level, the synchronisation types include the following:

1. **Synchronous:** All components of a distributed system are coordinated in time (as lock step or rounds) to be synchronised with each other. Causality is explicitly obtained. Examples include typical safety-critical systems such as aircraft fly-by-wire control where predictability and guaranteed real-time responsiveness is desired.
2. **Asynchronous:** Separate entities take steps in arbitrary order and operate at different speeds. The ordering of events needs to be ensured through collective interactions. Typical examples are transactional systems, databases, web crawlers, etc.
3. **Partially synchronous:** Some restrictions apply on ordering of actions but no lock-step synchronisation is present. Typical examples are SCADA control systems or high-value transactional stock systems where timeliness has implications on the service correctness.

4.2 Reliable and Secure Group Communication

Group communication addresses the communication schema available to ensure reliable delivery of messages across the distributed entities. These can be simple point-to-point direct messaging supported by appropriate acknowledgements (ACKS and NACKS) for reliable delivery. Alternately, reliable and secure multicast (atomic, best-effort, regular, uniform, logged, stubborn, probabilistic, causal, etc.) to provide redundant channels or ordering of messages can be used along with the more sophisticated publish-subscribe forms of group communication [2, 3]. In these approaches, the channels and messages can be encrypted or cryptographically signed though this entails higher transmission and processing overheads. The range of credential management, symmetric/asymmetric cryptography techniques, PKI cryptosystems, secure key distribution [40] also fall in this category. The reader is referred to [2, 1, 3, 41] for a comprehensive coverage of group communication primitives.

4.3 Coordination Properties

The utility of a distributed system comes from a coordinated orchestration of the dispersed resources to yield a collectively meaningful capability. Prior to discussing the variety of commonly used coordination schemas in Section 4.4, we first present the base definitions of *Consensus*, *Group Membership* and *Consistency*.

Consensus

Informally, consensus pertains to achieving an agreement on values. For example, the values could be data or process IDs. Consensus requires the following properties to hold:

1. **Agreement:** All good processes agree on the same value.
2. **Validity:** The agreed upon value is a good/valid value.
3. **Termination:** A decision is eventually achieved.

The specific type of consensus depends upon the semantics of the faults (*crash*, *omission*, *Byzantine*, etc.) to be addressed. The faults types are discussed in Section 5.

Group Membership and Consistency:

Membership is a key "service" property in distributed systems that determines the set of constituent resources and also the nature of the agreement achieved on the set of valid participants (static, dynamic, quorum membership) and the data. From a security perspective, this often relates to the integrity property for the service. Consistency has varied nuances and the prominent types are listed below with fuller details presented in [8, 1, 2, 3, 42, 41]. Note that the underlying assumption is always that the constituent processes can be modelled as deterministic state machines. That is, performing a specific sequence of actions always leads to the same state.

- **Strong consistency models:** In these models the participants must agree on one consistent order of actions to take. Hence, the processes are guaranteed to reach a consistent state under the assumption of determinism.

  1. *Strict Consistency:* In strict consistency there are no constraints on the observed order of actions as long as it is consistent across all the participants.
  2. *Linearisability:* The linearisability model is essentially strict consistency with the additional constraint that the observed order of actions corresponds to their real time order.

  Strong consistency models are widely used in high risk contexts where any inconsistencies in the data may lead to dire consequences. In these situations, consistency is more valued than availability and enforcing strong consistency constraints results in more delays in the systems due to the frequent synchronisation. Traditional relational database systems such as MySQL [43] or Microsoft's SQL Server [44] but also modern NoSQL databases such as MongoDB [45] or Google's Chubby lock service [46] are popular examples that implement these strong consistency models.

- **Weak Consistency Models:** In these models, the participants do not necessarily observe the same order of actions. This can lead to inconsistent states depending on the nature of the additional constraints that the observed orders have to satisfy. Naturally, this can lead to inconsistent states that can be dealt with through conflict resolution mechanisms [47].

  1. *Sequential Consistency:* Sequential consistency is met if the order in which the actions are performed by a certain process corresponds to their original order. In order words, the sequential execution order of every process is preserved.
  2. *Causal Consistency:* Causal consistency is achieved by categorising actions into those causally related/dependent and those that are not. In this case only the order of causally related actions has to be preserved. Two events are causally related if they both access the same data object and at least one of them is a write event.

3. *Eventual Consistency:* In eventual consistency there are no special constraints that have to be satisfied by the order of observer actions. The idea behind this concept is that the participants will *eventually* converge to a consistent state either by observing equivalent orders of actions or by resorting to *costly* conflict resolution mechanisms.

Systems with weaker consistency models became popular with the advent of the Internet where wide scale web servers had to accommodate a large number of users. To achieve this, such systems sacrifice strong consistency guarantees to achieve higher availability for their user base. Systems like Amazon's Dynamo [48], Facebook's Cassandra [49] are widely known examples of systems with weak consistency guarantees.

4.4 Replication Management and Coordination Schema: The Basis Behind Attack Mitigation

A fundamental challenge for developing reliable distributed systems is to support the cooperation of the dispersed entities required to execute a common task, even when some of these entities, or the communication across them, fails. There is a need to ensure ordering of the service actions and to avoid partitions of the distributed resources in order to result in an overall "coordinated" group of resources.

The state machine replication or state machine approach [50] is a general method for implementing a fault-tolerant service by replicating servers and coordinating client interactions with server replicas. The approach also provides a framework for understanding and designing replication management protocols. The essential system abstraction is that of a state machine such that the outputs of the state machine are fully determined by the sequence of requests it processes independent of time or other activity in the system. Replication can be active, semi-active, passive, or lazy [3].

It should be noted that ideally one would like to collectively attain high availability, consistency and also full coordination to eliminate any partitioning of the set of distributed resources. However, the CAP assertion comes into play as:

CAP

Any network shared data system (e.g. Web) can provide only 2 of the 3 possible properties [51] as:

1. **Consistency (C):** equivalent to having a single up-to-date copy of the data, i.e., each server returns the right response to each request.
2. **Availability (A):** of the data where each request eventually receives a response.
3. **Partition (P):** Network partition tolerance such that servers cannot get partitioned into non-communicating groups.

Naturally, security attacks attempt to compromise these elements of CAP.

Replication and Coordination

In order to provide coherent and consistent behaviour (in value and order), distributed resources use various types of replica management, i.e., the coordination schema. This is a key coordination mechanism that characterises the functionality of any distributed system. The factors determining the specific mechanism depend on the type of system synchronisation model, the type of group communication and especially the nature of the perturbations (faults or attacks) being considered. The mechanisms can be simple voting or leader election processes (e.g., Ring Algorithms, Bully) or more complex consensus approaches to deal with crashes or Byzantine[5] behaviour. The commit

[5]Byzantine behaviour happens when an entity/attacker sends different (albeit valid) information to different recipients.

protocols for database transactions are relevant here as are the schemes for credential management and PKI infrastructures providing verified access control. We briefly describe a set of widely used schema, and the reader is referred to [8, 2, 1] for complete coverage. Authorisation and Authentication in distributed systems are also discussed in the AAA KA.

Paxos

To avoid the situation of distributed entities conducting uncoordinated actions or failing to respond, Paxos [52], a group of implicit leader-election protocols for solving consensus in an asynchronous setup, has been developed. Paxos solves the consensus problem by giving all the participants the possibility to propose a value to agree upon in an initial phase. In the second phase, if a majority agrees on a certain value, the process that had proposed the value implicitly becomes the leader, and agreement is achieved. The same process is repeated for the next value to achieve consensus on a sequence of values.

The protocol is known not to provide liveness only under very specific circumstances as described in [52]. In this case, processes continue to propose values indefinitely and remain blocked in the initial phase as no majority can be formed and progress is never made. However, this situation rarely occurs in practice and Paxos remains one of most widely used coordination protocols.

Since only a majority is necessary in the second phase to reach consensus, the protocol is additionally tolerant to crashes even in the case of recovery. This is remarkable since, as long as the majority of the processes has not failed, consensus can be reached. The paper [53] is an excellent read of the experiences of implementing Paxos at Google for the Chubby file system.

While there exists a variety of implementations of the Paxos protocol, it is notoriously known for being hard to implement and build middleware upon it due to its inherent complexity. For this purpose, RAFT, a protocol similar to Paxos that provides the same guarantees, has been proposed. RAFT has recently gained in popularity due to its simpler design. Paper [54] explains the motivation behind the development of the RAFT protocol and how it works by comparing it with Paxos.

Byzantine Fault Tolerance (BFT)

Attacks and other deliberate disruptions do not necessarily follow the semantics of benign omissions, timing or crashes. In order to tolerate arbitrarily malicious behavior, Byzantine Fault Tolerant (BFT) protocols use coordinated replication to guarantee the correct execution of operations as long as at most a third of processes is compromised and exhibits arbitrary (i.e., Byzantine, cf. Section 5) behavior.

In BFT, processes exchange the values they have received from each other in rounds. The number of rounds necessary to reach consensus is determined by the number of compromised participants there are in the system [55]. Note that since the protocol operates in rounds, it is classified as a synchronous coordination protocol. It has been shown in [56] as the FLP impossibility result that it is impossible to reach consensus in the case of asynchronous communication. Due to the necessity of synchronous communication and the rather higher overhead of message exchange required to deal with Byzantine failures, BFT protocols are applied mostly in specific critical applications. However, there are multiple ongoing attempts for practical BFT optimisations by strengthening some basic assumptions on synchronisation, determinism, and number of compromises [57, 58, 59]. The Google File System (Chubby) and Amazon Web Services (AWS) implement Paxos and also partial BFT functionality. It is also important to emphasize that BFT is expensive not only for the message complexity over the number of rounds needed. It is also expensive for the number of nodes needed ($> 3f$) to handle f malicious failures, i.e., f being the number of nodes controlled by an adversary. The generalisation of adversarial structures to quorum systems is discussed in [41].

From a security viewpoint, for its ability to tolerate arbitrary malicious behaviors, the BFT protocols constitute an appealing building block for the construction of intrusion tolerant systems. It is worth

making the observation that these protocols consider the number of compromised entities. When faced with a malicious attacker identical replicas are not sufficient because they exhibit the same vulnerabilities. A malicious adversary who can compromise one replica can easily compromise the others if they are identical. Replication and diversity (or distinct protection methodologies) are needed. We refer the reader to the discussions in [55, 60, 61, 8, 1, 2, 41].

Commit Protocols

A number of applications, e.g., databases, require ordering across replicated data or operations where either all participants agree on conducting the same correct result (i.e., commit) or do nothing – the atomicity property. Hence, as a specialised form of consensus, a distributed coordinator directed algorithm is required to coordinate all the processes that participate in a distributed atomic transaction on whether to commit or abort (roll back) the transaction.

The two-phase commit protocol (2PC) is a straightforward example of such atomic commitment protocols. The protocol proceeds with a broadcast query from a leader to all the clients to commit. This is followed by an acknowledgment (commit or abort) from each client. On receiving all responses, the leader notifies all clients on an atomic decision to either commit or abort [2, 4, 5]. The protocol achieves its goal even in many cases of failure (involving either process, network node, or communication failures among others), and is thus widely used. An approach based on logging protocol states is used to support recovery. The classical 2PC protocol provides limited support for the coordinator failure that can lead to inconsistencies.

To solve this problem the three-phase commit (3PC) protocol has been developed. The 3PC protocol is essentially an extension of the BFT protocol and adds a third communication phase to assist the leader with the decision for an abort. This entails a higher messaging and logging overhead to support recovery. While 3PC is a more robust protocol compared to BFT, it is not widely used due to the messaging overhead and its sensitivity to network partitioning (i.e., the P in CAP). In practice, systems use either BFT for its simplicity or the Paxos protocol for its robustness.

5 Distributed Systems: Coordination Classes and Attackability

[8, c3][1, c5,c6][2, c19] [3, c18][19, c3]

The General Class of Disruptions

The attack surface [19, 62] in distributed systems involves the disruption of the resources, communication, interfaces, and/or data that either impairs the resource availability or disrupts the communication layer interconnecting the resources to impact Confidentiality, Availability, or Integrity of the overall system and its services. The disruptions can be from improper design, arising from operational conditions or deliberate attacks. Resource compromises or disruptions form the basic attack targets. However, the functionality of a distributed system emerges from the interactions across the distributed resources. As referenced in Section 1.2, the resources and services (including replication management) in a distributed system are primarily linked via communication infrastructures. These span the range of direct message exchanges or via middleware architectures such as pub-sub or event based triggering among others.

A number of varied terminologies exist to cover the range of operational and deliberate perturbations from crashes, omissions, timing, value disruptions, spoofing, viruses, trapdoors, and many others. We refer the reader to [20] for a comprehensive discussion on the topic.

As the distributed systems primarily rely on message passing for both data transportation and coordination, we group the perturbations at the level of message delivery[6]. The term “perturbation or

[6]The provisioning of message integrity by techniques such as coding, cryptographic primitives, message acknowledgements, retries, secure group communication, etc. are discussed in [2, 3] and the KA on Cryptography.

disruption" is deliberately used as the anomalous operation can result from operational issues (dependability) or from a malicious intent (security). The manifestation of these perturbations on the system operations results in deviations from the specified behavior of the system. Complementing the vulnerabilities mentioned in Section 1.2 of access control, data distribution, interfaces, the communication level perturbations can be broadly grouped as:

1. **Timing Based**: This spans the omission of messages, early, delayed, or out-of-order messaging. Crashes and denial-of-service also fall in this group as they typically manifest as disruptions of the proper temporal delivery of messages by obstructing access to the communication channels or resources.

2. **Value/Information Based:** Spoofing attacks, mimicking, duplication, information leakage such as a Covert Channel Attack or Side Channel Attack, and content manipulation attacks broadly fall in this category. The manipulation of the content of messages manifests as Byzantine behavior. This attack is only viable if a set of resources use the exchange messages to build their global view of the system. A malicious entity can send deliberately modulated information (e.g., a mixture of correct and incorrect values) to different groups of resources to result in partitions of system state views. Thus, based on different values received by different nodes, the individual nodes are unable to constitute a "consistent" and correct view of the system state. The degree of breach of consistency (strong – full agreement by all on value and order – weak, partial, eventual) constitutes the degree of disruption. The nature of the underlying transactional service (e.g., distributed ledgers in Blockchains) determines the type of breach of the functionality. Relating to the groups of vulnerabilities, a Byzantine attack can abuse access control, message delivery and coordination services, or the data itself (viruses, compromised mobile code, worms) to compromise the system.

It should be noted that a perturbation also includes the property of persistence, i.e., the duration of a perturbation can be transient, episodic, intermittent, or permanent in nature. Furthermore, attacks often entail multiple simultaneous occurrences that involve a combination of timing, value, persistence, and dispersed locations, potentially due to collusion between multiple attacking entities.

Attacks and Implications

On this general background, we now detail the two prominent classes of distributed systems as based on the coordination schema (resource- and service-coordination). This will also form the system grouping for considering the security manifestations of attacks.

We use the classical CIA (Confidentiality, Integrity, and Availability) terminology though the implications of these terms often differ according to the type of system and services. For each class, the specification of its functionality determines the type of attack and the resultant compromise that detrimentally affects the delivery of services.

As mentioned in Section 1.2, the threat surfaces of a distributed system comprise attacks on the resources, admission control, the communication architectures, the coordination mechanisms, and the data. Similarly, attacks aim to subvert the assumptions behind the functionality of resources, the services, and the underlying coordination schema.

In the following subsection, we enumerate some attack scenarios for the resources/infrastructure and services/application classes of coordination. Given the immense diversity of types of resource and services based distributed systems, the purpose of these examples is only to illustrate some potential scenarios. It is also worth highlighting that often a resource attack does not harm the resource *per se* but primarily affects the service executing on the resource.

5.1 The Resource Coordination Class – Infrastructure View

This class of "virtualised resource access" primarily deals with the coordination of a group of computing and communication resources to provide an ensemble of highly-available, highly-reliable "platform" of diverse shared resources to the user. This is an infrastructure (vs applications) view where the user specifies the operational requirements for the desired service (e.g., computational capabilities, number of Virtual Machines (VMs), storage, bandwidth constraints, etc.) but is agnostic to the actual mechanisms providing the on-demand access to the resources, scalability, physical characteristics, and geo-location/distribution of the underlying resources.

Overall, the key characteristic of this coordination model is the provisioning of high-reliability, high-availability access to resources. The basic resource replication simply provides a pool of resources to support high-availability access. However, the resource replication schema provides only the "capabilities" to support the services executing on it. Integrity is relevant corresponding to the service specifications. For instance, VMs need to provide the specified level of isolation without information leakage. Similarly, a web server is typically replicated across machines both for reliability and for low-latency localised geo-dispersed access. Each replicated server has the same set of data, and any time the data is updated, a copy is updated across the replicated servers to provide consistency on data. It is the nature of the service (as executing on the resources platform) that determines the type of desired coordination, perhaps as consistency (strong, weak, eventual, causal). This will be the basis of the Service Coordination class discussed later on.

We briefly present the Cloud and Client-Server models that constitute prominent examples of the class of distributed resources.

The Cloud Model

The Cloud, in all its manifestations, is representative of the resource coordination model as essentially a "resources platform" for services to execute on. There are multiple types of Clouds offering varied types of services ranging across emphasis on high-performance, low-latency access or high-availability amongst many other properties. It is the specific resource coordination schema dictated by the specifications of the desired services based on which the Cloud "platform" provides structured access to the Cloud resources. The chosen coordination schema correspondingly supports the type of desired capabilities, for example, access to specialised computing resources and/or resource containers such as physical or virtual machines each offering differing isolation guarantees across the containers. The user specified services execute on the Cloud resources, which are managed by the Cloud service provider. The coordination schema, as a centralised or distributed resource manager, handles the mapping and scheduling of tasks to resources, invoking VMs, health monitoring of resources, fault-handling of failed resources such that the user transparently obtains sustained access to the resources as per the contractual Service Level Agreements (SLAs) specified on the Cloud resources. The ENISA [63], NIST [64], and ISO [65] specifications of IaaS and PaaS[7] are representations of "resources/platforms/infrastructures supporting the services". The multitude of Cloud models, architectures, and services existing in practice makes it difficult to project a single notion of Cloud security. Each specific resource coordination model is characterized by the types of resource types in the Cloud model, the type of computing architecture as well as the desired functionalities within the Cloud. These include, as a non-exhaustive list, the desired types of resource fault handling, the chosen approach for handling of service bursts, the type of schemas implemented for resource federation and migration, for task orchestration, scheduling, the desired degree of concurrent access, the supported levels of multi-tenancy etc.

However, from a security perspective, it is useful to de-construct the Cloud into its architectural and functional components that result in the Cloud's attack surface to consider. Analogous to the infrastructure view of a data center being an aggregation of computing and storage resources, the

[7] IaaS: Infrastructure as a Service; PaaS: Platform as a Service

Cloud is an aggregation of geo-dispersed resources that are available on-demand to the user. The user has resource-location and resource-composition agnostic transparent access to highly-scalable, highly-available, highly-reliable resource and service virtualisation. The user specifies the operational attributes of interest (termed as Service Level Objectives) as (a) performance specifications, (b) reliability, (c) replication and isolation characteristics as types and number of VMs, (d) latency, (e) security as the level/degree of encryption and other mechanisms at the computing or communication level and (f) cost parameters for delivery or non-delivery of services in the form of contracts known as Service Level Agreements. The exact composition of the resources, their location or the mechanisms collating the aggregated resources is *transparent* to the user. The functional blocks of the Cloud include authentication, access control, admission control, resource brokering, VM invocation, schedulers, monitors, reconfiguration mechanisms, load balancers, communication infrastructures, user interfaces, storage, and many other functions under the PaaS and IaaS paradigms [65, 63, 64]. These functional blocks, the physical Cloud resources along with the interfaces across them directly constitute the attack surface of the Cloud.

The Client-Server Model

Resource groups where a set of dedicated entities (servers – service providers) provide a specified service (e.g., Web services – file system servers, name servers, databases, data miners, web crawlers, etc.) to a set of data consumers (clients). A communication infrastructure, such as the public Internet, a local network, or a combination thereof, links the servers to the clients. This can be monolithic, layered, or hierarchical. Both servers and clients are replicated to either provide a characteristic collective distributed service or for fault tolerance. Note that we are referring to Client-Server architecture as a resources platform or infrastructure and not the Client-Server services per se. The functionality of a Client-Server infrastructure is derived from the specifications of the services using the Client-Server model and from the requisite coordination schema underlying it.

Attackability Implications (and Mitigation Approaches) on Resource Coordination

We now outline some example scenarios for the Cloud though they analogously apply to the Client-Server and other resource models as well. The reader is referred to [66, 67] for an insightful discussion relating security and functionality issues in the Cloud.

- *Compromise of Resources:* Such attacks impact the Availability of the basic resources.

Mitigation: Protection can be obtained by using access control schemes (including Firewalls) to limit external access to services and network resources. Authorisation processes are set up for granting of rights along with access control mechanisms that verify the actual rights of access [68]. Other approaches to resource protection include the sandboxing resources or having a tamper-resistant Trusted Computing Base (TCB) that conducts coordination handling [2, 3] and enforces resource accesses. While the resource class primarily considers attacks on the infrastructure, data at-rest or in-motion (as in a data storage facility) can also be considered as a resource. Consequently, it can be protected using techniques such as encryption. As the specification of a distributed service includes the specification of both normal and anomalous behavior on the use of the data providing the service, this protection is considered under the services class.

Other manifestation of resource attacks, including on communication channels, aim to partition resources (and overlying services). The implication here is on Availability for the resources and on Integrity for the services.

- *Compromise of Access/Admission Control:* This comprises the broad categories of Masquerading, Spoofing, and ID management attacks. The implication on the resources is on Availability, though both the Integrity and Confidentiality of the data/service are affected. In case of a DoS attack, the consequence is on resource Availability.

Mitigation: Intrusion Detection Schemes (IDS) constitute typical mitigation approaches. These are

complemented by periodic or random ID authentication queries. The periodic checking of system state is used to establish the sanity of IDs.

- *Compromise of VM:* The typical manifestation is of information leakage from the VM via a Covert Channel Attack or Side Channel Attack or similar attacks. The consequence is the violation of Integrity and Confidentiality of the services provisioned by the VM.

Mitigation: A variety of schemes for VM's protection are detailed in [39] (also see the Operating Systems and Virtualisation KA). There are three aspects to be considered here as the detection of leakage, the system level where the leakage transpires, and the handling of leakage. Taint analysis is a powerful technique for data level detection. As covert/side-channel attacks often happen at the hardware level and are influenced by the schedulers, the use of detectors employing hardware performance counters is a generally used technique as advocated in [69]. System level handling of VM compromises often starts from the level of tightening the specification of trust assumptions and validating them being upheld using analytical, formal, or experimental stress techniques. Hypervisors are commonly used for the enforcement of VM operations.

- *Compromise of Scheduler:* There are two manifestations of such attacks. When the scheduler is affected and this results in an anomalous task or resource allocation, such a deviation (on an incorrect resource allocation) can be detected through Access Control. In the case of a malicious takeover of the scheduler, the likely resultant inconsistencies across the system state or resource-task bindings can be filtered by the coordination schema whose job is to maintain a consistent state. Such attacks typically impact Availability and Integrity. Confidentiality is not breached.

Mitigation: As mentioned in the attack description, Access Control and coordination constructs are used to check the consistency of the system state for any observed mis-match to the legitimate or allowed set of resource allocations. This can be used identify corruptions of the scheduler.

- *Compromise of Broker:* This occurrence, within a Cloud resource manager/broker or an inter-Cloud broker, primarily impacts resource Availability.

Mitigation: Approaches similar to scheduler compromise mitigation are used here. If backup brokers are part of the design, that is a typical fall back, otherwise, system stops are often the solution.

- *Compromise on Communication:* As communication is a core functionality to achieve resource coordination, this has strong implications on the resources to stay coordinated and directly impacts Availability. The consequent inability to support replication, resource to task allocation, etc. fundamentally compromises the functionality of the system.

Mitigation: A variety of communication protection techniques are presented in the Network Security KA. These include retries, ACK/NACK based schemes, cryptographically secured channels among others.

- *Compromise on Monitoring and Accounting:* With incorrect information on the state of the system and/or services, this can lead to compromise of Confidentiality, Integrity, and Availability.

Mitigation: State consistency schemes are the typical mechanism utilised here. It is worth mentioning that the replication and coordination concepts presented in Sections 4 and 4.4 form the basis of the mitigation approaches. The very purpose of the replication management is to obtain consistent system states to circumvent disruptions.

5.2 The Services Coordination Class – Applications View

The service coordination model focuses on the specific characteristics of the services that determine the degree/type of coordination relevant to supporting that service. For example, a database hosted on a Cloud necessarily requires the provision of integrity in the form of ACID[8] properties along with

[8] A stands for atomic, C for consistent, I for isolated and D for durable.

liveness. Distributed storage, such as KVS (Key Value Store) or transactional database services, may require varied levels of consistency or linearisability where the desired level of integrity may depend on the level of data-access latency feasible in the system. The broad class of Web services to include Web crawlers and search engines may require weak or partial consistency as per CAP. On the other hand, Blockchains or ledger queries, that provide distributed crypto based consensus, have strong consistency (and traceable auditing) as a key requirement with lesser demands on latency. Thus, it is the specification of the service (KVS, Database, Blockchain) that determines the nature of the coordination schema for the distributed resources platform.

We present some characteristic examples of the services class as:

Web Services: These cover the spectrum of data mining, web crawlers, information servers, support for e-transactions, etc. This is a fairly broad, and generic, category, which encompasses a wide variety of services. It is useful to note that many of these services utilise the Client-Server paradigm though our interest here is at the services level.

Key Distribution: This is a broad class of (Authorisation & Authentication) services such as Kerberos, PKI, etc. Such services typically enable authentication (either proving server authenticity to a client, or mutually authenticating both client and server) over insecure networks, based on various cryptographic protocols. Authentication services commonly act as trusted third party for interacting entities in a distributed system. For further details see the AAA KA.

Storage/KVS

This is a diverse set of services starting from register level distributed read-writes that entail strong consistency with very low latency. Another general model is Key Value Stores (KVS) where data is accessed via keys/pointers/maps with simple read, write, delete types of semantics. In KVS, the data is represented as a collection of key-value pairs, such that each possible key appears at most once in the collection with a focus on fast access times (up to a constant access time). The key-value model is one of the simplest non-trivial data models, and richer data models are often implemented as extensions with specified properties. For example, an ordered model can be developed that maintains the keys in a lexicographic order to efficiently retrieve selective key ranges. Key-value stores can use consistency models ranging from eventual consistency to strict consistency. The security issues requires dealing with data-at-rest (static storage) and data-in-transit (dynamic R/W ops).

Transactional Services, Databases

This is a wide class of services covering databases and general transactional services (retrieval, informational data mining, banking and stock transactions, etc.). The requirements are consistency as in banking where all the debit and credit transactions are (strongly or weakly) serializable for consistency. More generally, a database adheres to all of the stipulated ACID properties.

On the other hand, a number of data mining and information lookup transactions only require weaker nuances of consistency. For example, an information lookup process can work with physically partitioned data centers resulting in stale or inconsistent information as long as they are eventually reconcilable within some specification of the service requirements. The specification of the type and degree of perturbations and level of consistency the services are designed to be resilient to determines the specific coordination schema to use. Additionally, in the case of weaker consistency models, the user is required to deal with any stale data that might have been retrieved from the database.

Blockchains/Cryptocurrencies

The concept of a ledger provides for consistent bookkeeping on transactions. This is problematic to achieve in a distributed system where the participating entities do not trust each other and are potentially untrustworthy. Blockchains provide a decentralised, distributed, and public ledger that is used to record transactions across many computers so that the record cannot be altered retroactively without also altering all subsequent blocks. Such alterations require the consensus of the network and

can therefore not be performed unilaterally by an attacker. This also allows the participants to verify and audit transactions inexpensively. Blockchains form the foundation for numerous cryptocurrencies, most notable Bitcoin.

In technical terms, a Blockchain is a list of records or blocks. The aforementioned properties arise from the fact that each block incorporates a cryptographic hash of the previous block and a timestamp. If a block in the chain is altered without also altering all subsequent blocks, the hash of the following block will no longer match, making the tampering on the Blockchain detectable.

When used as distributed ledgers, Blockchains are typically managed by peer-to-peer networks. Peers in such a network participate in a protocol for validating newly submitted blocks. Blockchains are also examples of widely deployed systems exhibiting high tolerance to Byzantine failures.

The generic Blockchain concept allows participation by any entity (permission-less systems, public blockhains) and does not include any access restrictions. This is the case for the blockchains underlying many widely used cryptocurrencies such as Bitcoin. However, a more restrictive participation model (permissioned systems, private blockchains) is also possible, where a "validating authority" grants permission for participation.

In order to deter denial of service attacks and other service abuses such as spam on a network, the concept of Proof-of-Work (PoW) (i.e., spending processing time to perform computationally expensive tasks) is specified as a requirement for participation. This is effective as a means of preventing service abuses such as spam since the required work is typically hard to perform but easy to verify, leading to asymmetric requirements for service requester and provider. However, PoW schemes also lead to high energy usage and, depending on the chosen work requirement, may lead to unreasonably high barriers of entry. This is the case, for instance, in certain cryptocurrencies, where meaningful participation requires custom hardware designed for the specific type of work required. To avoid these shortcomings, alternative approaches relying on Proof-of-Stake (PoS) are in development but not as mature as PoW-based schemes and not widely deployed.

A comprehensive discussion on Blockchain issues appears in [70, 71]. As a note, Blockchains represent an interesting combination of decentralised resources using the P2P model for the resource coordination and the coordination schema of consensus for its service functionality.

Overall, service integrity, in terms of consensus as supported by requisite liveness, is the key characteristic of the service coordination model. This contrasts with the resource coordination class where resource accessibility and availability were the dominant drivers/considerations.

Attackability Implications (and Mitigation Approaches) on Service Coordination

The services and applications constitute a very broad class to cover, both for the type of attacks and the diversity of services where the functional specification of the service determines the type and degree of the impact on security. In most cases the breach on Integrity, along with on Confidentiality, is the first class impact with impact on Availability following as a consequence. Some examples of breaches for the coordination schema and service types are mentioned below.

Note: The mitigation schemes applicable here are the same as described in Section 5.1 that essentially result from the basic replication management and coordination concepts presented in Sections 4 and 4.4. The very purpose of replication based coordination, at the resource or the service level, is to prevent compromises by discrete attacks up to the threshold of severity type and the number of disruptions the replication schema is designed to handle.

Compromise of Key distribution in PKI: The authentication processes supporting the distribution of public keys is compromised affecting service Integrity and Confidentiality.

Compromise of Data at Rest: This is analogous to the breach of resources in the resource coordination model as applicable to storage systems.

Compromise of Data in Motion: This has varied consistency and latency consequences that compromise the Integrity depending on the specifications of the services. We present a very simplistic enumeration using transactions classes as:

Short transactions: (Storage/KVS, etc.) The major driver for this class is both consistency and low latency (e.g., linearisability). As both liveness and safety are violated, the Integrity of the transaction is compromised. It is worth noting that a DoS attack may not affect consistency. However, as latency is affected, the service Integrity is lost.

Large transactions: Ledgers (Blockchain, etc.) lie in this category where, although latency is important, it is the Integrity (as defined by the consistency of the ledger) that is the primary property to preserve. As Ledgers constitute a popular service, we discuss it to illustrate aspects of both attack surfaces and assumptions.

To recapitulate from Section 5.2, Blockchains constitute a ledger of information that is dispersed across a distributed system. Blockchains ensure the security of data by not providing a single point of attack. The ledger is stored in multiple copies on a network of computers. Each time an authorised participant (for example in a permissioned system) submits a transaction to the ledger, the other participants conduct checks to ensure that the transaction is valid, and such valid transactions (as blocks) are added to the ledger chain. Consensus ensures a consistent view of the sequence of transactions and the collated outcome. The cryptographic basis of the hash, on each block, is expected to avoid tampering, and the Proof of Work notion is designed to mitigate the effect of DoS attacks.

What makes this system theoretically tamper proof are two aspects: (a) an unforgeable cryptographic hash linking the blocks, and (b) attack-resilient consensus by which the distributed participants agree on a shared history of transactions.

Compromising these involves the compromise of stored cryptographic keys and the hash. While theoretically safe, such systems may turn out to be vulnerable to emergent technologies such as quantum computing. Moreover, while Proof of Work requirements (i.e., "to demonstrate" a greater than 50% participant agreement) can make collusion attacks prohibitively expensive in sufficiently large systems, they can be feasible on systems with fewer participants.

Similarly, the consensus property can be compromised via an Eclipse attack [72] for Bitcoin, and also in general cases where there exists the potential to trick nodes into wasting computing power. Nodes on the Blockchain must remain in constant communication in order to compare data. An attacker that can take control of a node's communication and spoof other nodes into accepting false data to result in wasted computing or confirming fake transactions can potentially breach consensus. The work in [71] provides useful reading on such compromises.

Mixed transactions: As implied in the label, this combines short and large transactions. The security implications depend on the type of services. As an example, we outline two service groups, namely:

- E-commerce supporting transactions: The core requirements here are ACID properties that entail strong consistency and no partitions. Any compromises affect the Integrity of the service.

- Informational systems: Services such as Webcrawlers, Data Retrieval for applications such as Uber or informational queries for shopping can handle (both network and data) partitions of data to operate on stale cached data. The attack may lead to redundant computations on the searches or slightly stale information but Integrity is not violated as long as the semantics of Weak, Relaxed, or Eventual consistency, as applicable for the service specification, are sustained. Also informational queries have mixed latency requirements. For example, the small latency within a local data center and higher-tolerable latency across geo-dispersed data centers may define the degree of attack tolerance until both Availability and Integrity are compromised.

CONCLUSIONS

The intent of this chapter has been to outline how distributed systems work, and how the mechanisms supporting the operations of such systems open security issues in them. Very often the expectation is that classical security techniques will directly apply in a distributed systems context as well. However, this is often not the case and the better one understands the conceptual basis of a distributed system, the better one can understand and provide for its security. The KA discussed the functional categorisation of distributed systems into two major classes: decentralised and coordinated control. The operations for each class were elaborated leading to the security implications resulting from the different specifics underlying distributed systems.

CROSS-REFERENCE OF TOPICS VS REFERENCE MATERIAL

	Lynch [8]	Rachid [1]	Birman [2]	PJV [3]	Snyder [19]
1 Classes of Distributed Systems and Vulnerabilities		c2	c5	c18	
2 Distributed Systems: Decentralised P2P Models	c11,c12		c25		
3 Distributed Systems: Attacking P2P Systems				c16	c5
4 Distributed Systems: Coordinated Resource Clustering	c5-7, c12, c25	c3	c5,c14	c16, c17, c19	
5 Distributed Systems: Coordination Classes and Attackability	c3	c5-6	c19	c18	c3

FURTHER READING

The following books are recommended for a deeper coverage of the distributed system and security concepts.

5.2.1 Distributed Algorithms Concepts

Lynch [8] — The book lays out the essential concepts of distributed systems. The focus is on synchronisation and consensus though it provides a comprehensive and mathematically rigorous coverage of distributed systems concepts from an algorithms viewpoint.

5.2.2 Reliable & Secure Distributed Programming

Cachin, Guerraoui, Rodrigues [1] — Coming from a distributed programming viewpoint, this is another rigorous book that covers both fault tolerance and security. It also provides an excellent coverage of cryptographic primitives. Although it predates the development of Ledgers, most of the concepts behind them are covered in this book.

5.2.3 Group Communication & Replication

Birman [2] — This is an excellent book that combines concepts with an emphasis on the actual development of distributed systems. The case studies provide valuable insights on practical issues and solutions. An insightful coverage of P2P systems also appears in this book.

5.2.4 Security Engineering

Anderson [6] — This book makes for excellent reading on the realisation of distributed system from a security perspective especially for naming services and multi-level security. The reader is also encouraged to read the texts [38, 7] that detail complementary coverage on CORBA and Web services.

5.2.5 Threat Modeling

Swiderski, Snyder [19] — The coverage is on the basics of threat modeling from a software life cycle and application security viewpoint. While not a distributed systems book, it still provides valuable insights on how threat modeling is conducted in practice.

REFERENCES

[1] C. Cachin, R. Guerraoui, and L. Rodrigues, *Introduction to Reliable and Secure Distrbuted Programming*. Springer, 2011.

[2] K. Birman, *Reliable Distributed Systems*. Springer, 2005.

[3] P. Verissimo and L. Rodriques, *Distributed Systems for System Architects*. Kluwer, 2001.
[4] A. Tannenbaum and M. Steen, "Distributed Systems: Principles & Paradigms," *Prentice Hall*, 2007.
[5] M. Steen and A. Tannenbaum, "Distributed Systems," *Prentice Hall*, 2017.
[6] R. Anderson, "Security Engineering," *Wiley*, 2008.
[7] B. Hartman, D. Flinn, and K. Beznosov, "Enterprise Security with EJB and CORBA," *Wiley*, 2001.
[8] N. Lynch, *Distributed Algorithms*. Morgan Kaufmann, 1996.
[9] P. Eugster, P. Felber, R. Guerraoui, and A.-M. Kermarrec, "The Many Faces of Publish/Subscribe," *ACM Computing Surveys*, vol. 35, no. 2, pp. 114–131, 2003.
[10] M. Ripeanu, "Peer-to-Peer Architecture Case Study: Gnutella Network," in *Peer-to-Peer Computing*, 2001, pp. 99–100.
[11] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, and S. Shenker, "Making Gnutella-like P2P Systems Scalable," in *Proc. of Applications, Technologies, Architectures, and Protocols for Computer Communications*, ser. SIGCOMM. ACM, 2003, pp. 407–418.
[12] I. Stoica et al., "Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications," *In Proc. SIGCOMM*, pp. 149 – 160, 2001.
[13] A. Rowstron and P. Druschel, "Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems," *Proc. Middleware*, pp. 329–350, 2001.
[14] B. Zhao, L. Huang, J. Stribling, S. Rhea, A. Joseph, and J. Kubiatowicz, "Tapestry: A Resilient Global-scale Overlay for Service Deployment," *IEEE Journal on Selected Areas in Communications*, vol. 22, no. 1, pp. 41–53, Jan 2004.
[15] P. Maymounkov and D. Mazières, "Kademlia: A Peer-to-Peer Information System Based on the XOR Metric," *In Proc. IPTPS*, pp. 53 – 65, 2002.
[16] BitTorrent, "BitTorrent Protocol Specification," http://www.bittorrent.org/beps/bep_0003.htm, Tech. Rep., 2008.
[17] B. Yang and H. Garcia-Molina, "Comparing Hybrid Peer-to-Peer Systems," *Proc. of VLDB*, pp. 561–570, 2001.
[18] L. Garcés-Erice, E. W. Biersack, K. W. Ross, P. Felber, and G. Urvoy-Keller, "Hierarchical Peer-To-Peer Systems," *Parallel Processing Letters*, vol. 13, no. 4, pp. 643–657, 2003.
[19] F. Swiderski and W. Snyder, *Threat Modeling*. Springer, 2003.
[20] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *Dependable and Secure Computing, IEEE Transactions on*, vol. 1, no. 1, pp. 11–33, Jan 2004.
[21] C. Esposito and M. Ciampi, "On Security in Publish/Subscribe Services: A Survey," *IEEE Communication Surveys and Tutorials*, vol. 17, no. 2, 2015.
[22] A. Uzunov, "A Survey of Security Solutions for Distributed Publish/Subscribe Systems," *Computers and Security*, vol. 61, pp. 94–129, 2016.
[23] A. Walters, D. Zage, and C. Rotaru, "A Framework for Mitigating Attacks Against Measurement-Based Adaptation Mechanisms in Unstructured Multicast Overlay Networks," *IEEE/ACM Trans on Networking*, vol. 16, no. 6, pp. 1434–1446, Dec 2008.
[24] T. Isdal, M. Piatek, and A. Krishnamurthy, "Privacy Preserving P2P Data Sharing with Oneswarm," *SIGCOMM*, 2011.
[25] J. Seibert, X. Sun, C. Nita-Rotaru, and S. Rao, "Towards Securing Data Delivery in Peer-to-Peer Streaming," in *Proc. Communication Systems and Networks (COMSNETS)*, 2010, pp. 1–10.
[26] R. Barra de Almeida, J. Miranda Natif, A. Couto da Silva, and A. Borges Vieira, "Pollution and Whitewashing Attacks in a P2P Live Streaming System: Analysis and Counter-Attack," in *IEEE Intl. Conf on Communications (ICC)*, June 2013, pp. 2006–2010.
[27] J. Liang, N. Naoumov, and K. Ross, "The Index Poisoning Attack in P2P File Sharing Systems," in *INFOCOM*, 2006, pp. 1–12.
[28] N. Naoumov and K. Ross, "Exploiting P2P Systems for DDoS Attacks," in *ACM Proceedings of Scalable Information Systems*, 2006.

[29] D. Li, J. Wu, and Y. Cui, "Defending Against Buffer Map Cheating in DONet-Like P2P Streaming," *IEEE Trans. Multimedia*, vol. 11, no. 3, pp. 535–542, April 2009.
[30] J. R. Douceur, "The Sybil Attack," in *Intl. Workshop on Peer-to-Peer Systems*. Springer-Verlag, 2002, pp. 251–260.
[31] A. Singh et al., "Eclipse Attacks on Overlay Networks: Threats and Defenses," *Proc. INFOCOM*, pp. 1–12, 2006.
[32] F. DePaoli and L. Mariani, "Dependability in Peer-to-Peer Systems," *IEEE Internet Computing*, vol. 8, no. 4, pp. 54–61, July 2004.
[33] G. Gheorghe, R. Lo Cigno, and A. Montresor, "Security and Privacy Issues in P2P Streaming Systems: A Survey," *Peer-to-Peer Networking and Applications*, vol. 4, no. 2, pp. 75–91, 2011.
[34] G. Urdaneta, G. Pierre, and M. V. Steen, "A Survey of DHT Security Techniques," *ACM Comput. Surv.*, vol. 43, no. 2, pp. 8:1–8:49, 2011.
[35] Y.-K. Kwok, "Autonomic Peer-to-Peer Systems: Incentive and Security Issues," in *Autonomic Computing and Networking*, Y. Zhang, L. T. Yang, and M. K. Denko, Eds. Springer, 2009, pp. 205–236.
[36] S. Androutsellis-Theotokis and D. Spinellis, "A Survey of Peer-to-peer Content Distribution Technologies," *ACM Computing Surveys*, vol. 36, no. 4, pp. 335–371, Dec. 2004.
[37] D. Wallach, "A Survey of Peer-to-Peer Security Issues," in *Software Security - Theories and Systems*, ser. Lecture Notes in Computer Science. Springer, 2003, vol. 2609, pp. 42–57.
[38] B. Hartman, D. Flinn, and K. Beznosov, "Mastering Web Services Security," *Wiley*, 2003.
[39] D. Sgandurra and E. Lupu, "Evolution of Attacks, Threat Models, and Solutions for Virtualized Systems," *ACM Computing Surveys*, vol. 48, no. 3, pp. 114–131, 2016.
[40] M. Reiter, "How to Securely Replicate Servers," *ACM Trans. Programming Language Systems*, pp. vol. 16, 3, 986–1009, 1994.
[41] M. Vukolic, "Quorum Systems: Applications to Storage & Consensus," *Morgan Claypool*, 2012.
[42] P. Viotti and M. Vukolic, "Consistency in Non-Transactional Distributed Storage Systems," *ACM Computing Surveys*, vol. 49, no. 1, 2016.
[43] P. DuBois and M. Foreword By-Widenius, *MySQL*. New riders publishing, 1999.
[44] https://www.microsoft.com/en-us/sql-server.
[45] https://www.mongodb.com.
[46] M. Burrows, "The chubby lock service for loosely-coupled distributed systems," in *Proceedings of the 7th symposium on Operating systems design and implementation*. USENIX Association, 2006, pp. 335–350.
[47] Y. Saito and M. Shapiro, "Optimistic Replication," *ACM Computing Surveys (CSUR)*, vol. 37, no. 1, pp. 42–81, 2005.
[48] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, "Dynamo: amazon's highly available key-value store," in *ACM SIGOPS operating systems review*, vol. 41, no. 6. ACM, 2007, pp. 205–220.
[49] A. Lakshman and P. Malik, "Cassandra: a decentralized structured storage system," *ACM SIGOPS Operating Systems Review*, vol. 44, no. 2, pp. 35–40, 2010.
[50] F. Schneider, "Implementing Fault Tolerant Services Using the State Machine Approach: A Tutorial," *ACM Computing Surveys*, p. 23(4), 1990.
[51] E. Brewer, "CAP: Twelve Years Later: How the Rules Have Changed," *IEEE Computer*, pp. 23–29, Feb 2012.
[52] L. Lamport, "Paxos Made Simple," *ACM SIGACT News*, 2001.
[53] T. Chandra, R. Griesemer, and J. Redstone, "Paxos Made Live: An Engineering Perspective," *Proc. of ACM PODC*, 2007.
[54] D. Ongaro and J. Ousterhout, "In search of an understandable consensus algorithm," in *USENIX Annual Technical Conference (USENIX ATC)*, 2014, pp. 305–319.
[55] L. Lamport, R. Shostak, and M. Pease, "The Byzantine Generals Problem," *ACM Transactions on Programming Languages and Systems*, p. 4(3), 1982.

[56] M. J. Fischer, N. A. Lynch, and M. S. Paterson, “Impossibility of Distributed Consensus with One Faulty Process,” Yale Univ, Dept of CS, Tech. Rep., 1982.
[57] R. Kotla, L. Alvisi, M. Dahlin, A. Clement, and E. Wong, “Zyzzyva: Speculative Byzantine Fault Tolerance,” *ACM SIGOPS Operating Systems Review*, vol. 41, no. 6, pp. 45–58, 2007.
[58] M. Castro and B. Liskov, “Practical Byzantine Fault Tolerance and Proactive Recovery,” *ACM Transactions on Computer Systems (TOCS)*, vol. 20, no. 4, pp. 398–461, 2002.
[59] Y. Zhang, Z. Zheng, and M. R. Lyu, “BFTCloud: A Byzantine Fault Tolerance Framework for Voluntary-Resource Cloud Computing,” in *IEEE Conf. on Cloud Computing*, 2011, pp. 444–451.
[60] M. Castro and B. Liskov, “Practical Byzantine Faut Tolerance,” *Proc. of OSDI*, pp. 23–29, 1999.
[61] M. Platania, D. Obenshain, T. Tantillo, Y. Amir, and N. Suri, “On Choosing Server- or Client-Side Solutions for BFT,” *ACM Comput. Surv.*, vol. 48, no. 4, pp. 61:1–61:30, 2016.
[62] P. Manadhata and J. Wing, “An Attack Surface Metric,” *IEEE Trans Software Engineering*, vol. 37, no. 3, pp. 371–386, May 2011.
[63] European Network and Information Security Agency, “Survey and Analysis of Security Parameters in Cloud SLAs Across the European Public Sector,” 2014.
[64] National Institute of Standards and Technology, NIST 800-53v4, “Security and Privacy Controls for Federal Information Systems and Organizations,” 2014.
[65] International Organization for Standardization ISO-IEC 27002, “Guidelines on Information Security Controls for the Use of Cloud Computing Services Based on ISOIEC 2700,” 2014.
[66] A. Bessani et al, “Secure Storage in a Cloud-of-Clouds,” *ACM Eurosys*, pp. 31–46, 2011.
[67] G. Karame et al, “Reconciling Security and Functional Requirements in Multi-tenacy Clouds,” *Proc. of SCC*, 2017.
[68] B. Lampson, “Protection,” *Operating Systems Review*, pp. 8, 1, 18–24, 1974.
[69] T. Zhang, Y. Zhang, and R. B. Lee, “Cloudradar: A real-time side-channel attack detection system in clouds,” in *Proceedings of Research in Attacks, Intrusions, and Defenses - 19th International Symposium, RAID 2016*, 2016, pp. 118–140.
[70] E. Androulaki et al, “Hyperledger Fabric: A Distributed Operating System for Permissioned Bockchains,” *ACM Eurosys*, 2018.
[71] A. Gencer, S. Basu, I. Eyal, R. van Renesse, and E. Sirer, “Decentralization in Bit Coin and Ethereum Networks,” *Financial Cryptography and Data Security Conference*, 2018.
[72] E. Heilman, A. Kendler, A. Zohar, and S. Goldberg, “Eclipse Attacks on Bitcoin’s Peer-to-Peer Network,” *USENIX Security Symposium*, pp. 129–144, 2015.