Study Protocol

Research questions

RQ1	What are the smells indicating possible security violations in microservice-based applications?
RQ2	How to refactor microservice-based applications to mitigate the effects of security smells therein?

White Literature Search

Database	Search String	Results	
ACM Digital Library	microservice* AND secur* AND (smell OR antipattern OR "bad practice" OR pitfall OR refactor OR reeingeer OR restructure)	101	
DBLP	microservice security (smell | antipattern | bad practice | pitfall | refactor | reeingeer | restructure)	1	
IEEE Xplore	microservic* AND secur* AND (smell OR antipattern OR "bad practice" OR pitfall OR refactor OR reeingeer OR restructure)	4	
ISI Web of Science	ALL=(microservice*) AND ALL=(secur*) AND (ALL=smell OR ALL=antipattern OR ALL="bad practice" OR ALL=pitfall OR ALL=refactor OR ALL=reengineer OR ALL=restructure)	2	
Science Direct	microservice AND security AND (smell OR antipattern OR "bad practice" OR pitfall OR refactor OR reeingeer OR restructure)	68	
Scopus	microservice AND security AND (smell OR antipattern OR "bad practice" OR pitfall OR refactor OR reeingeer OR restructure)	129	
SpringerLink	microservic* AND secur* AND (smell OR antipattern OR "bad OR practice" OR pitfall OR refactor OR reengineer OR restructure)	629	
Google Scholar	(microservice) AND (security) AND (smell OR antipattern OR "bad practice" OR pitfall OR refactor OR reengineer OR restructure)	169	(restricted to scientific articles)

Grey Literature Search

Search Engines	(effort bounded: first 250 hits)		
Query	**Google**	**Bing**	**DuckDuckGo**
microservice security smell	250	250	250
microservice security antipattern	250	250	250
microservice security bad practice	250	250	250
microservice security pitfall	250	250	250
microservice security refactor	250	250	250
microservice security reenginer	250	250	250
microservice security restructure	250	250	250

Results 6353

Inclusion criteria

A study is selected if published between 01/01/2011 and 31/12/2020
A study is to be selected if it is written in English.
A study is to be selected if it qualifies as white literature, or as a blog post, whitepaper, industrial magazine publication, or video authored by a practitioner.

A study is to be selected if it focuses on microservices.
A study is to be selected if it focuses on security.

A study is selected if it presents at least one security smell possibly resulting in a violation of a security property defined by the ISO/IEC 25010 standard
A study is selected if it presents at least one refactoring for mitigating the effects of a security smell, even if the latter is not explicitly mentioned.

Coding Schema
type (white/grey)
sub-type [white] conference vs journal paper vs book chapter | [grey] blog post vs video vs whitepaper vs documentation
authors
title
publication venue: [white] bibliographic information | [grey] name of the blog/news (e.g., TechBeacon, Medium), video hosting service, …
year
URL/link
Search engine

Candidate Literature (Inclusion/Exclusion)

Authors	Title	Venue	Year	Type	Sub-Type	Search engine	Francisco	Jacopo	IRR	ALL	Link
Marco Troisi	8 best practices for microservices app sec	TechBeacon	2017	Grey	Blog post	Google	Included	Included	1	Included	https://techbeacon.com/app-dev-testing/8-best-practices-microservices-app-sec
~~Jack Mannino~~	~~Security In A Microservice World~~	~~OWASP~~	~~2017~~	~~Grey~~	~~Presentation (PPT)~~	~~Google~~	Excluded	Excluded	1	Excluded	~~https://owasp.org/www-pdf-archive/Microservice_Security.pdf~~
Jack Mannino	Security In The Land Of Microservices	AppSec EU 2017	2017	Grey	Video	Google	Included	Included	1	Included	https://www.youtube.com/watch?v=JRmWlLY8MGE
Zach Gardner	Security in the Microservices Paradigm	Dzone	2017	Grey	Blog post	Google	Excluded	Included	0	Included	https://dzone.com/articles/security-in-the-microservices-paradigm
~~Tim Leytens~~	~~API security in a microservices architecture~~	~~Medium~~	~~2019~~	~~Grey~~	~~Blog post~~	~~Google~~	Excluded	Excluded	1	Excluded	https://medium.com/@timleytens/api-security-in-a-microservices-architecture-2ef673e807c
Sumo Logic	Improving Security in Your Microservices Architecture	Sumo Logic	2019	Grey	Blog post	Google	Included	Included	1	Included	https://www.sumologic.com/insight/microservices-architecture-security/
~~Mick Knutson, Robert Winch, Peter Mularien~~	~~Spring Security: Secure your web applications, RESTful services~~	~~Packt~~	~~2017~~	~~Grey~~	~~Book~~	~~Google~~	Excluded	Excluded	1	Excluded	https://books.google.cl/books?id=MkBPDwAAQBAJ&pg=PA434&lpg=PA434&dq=microservice+restructur
Thribhuvan Krishnamurthy	Transition to Microservice Architecture - Challenges	Beingtechie	2018	Grey	Blog post	Google	Included	Included	1	Included	https://www.beingtechie.io/blog/transition-to-microservices-challenges
~~Stefano Di Paola~~	~~Microservices Security: Dos and Dont's~~	~~Minded Security~~	~~2018~~	~~Grey~~	~~Blog post~~	~~Google~~	Excluded	Included	0	Excluded	https://blog.mindedsecurity.com/2018/07/microservices-dos-and-donts.html
Vinay Sahni	Best Practices for Building a Microservice Architecture	Vinay Sahni	2019	Grey	Blog post	Google	Excluded	Included	0	Included	https://www.vinaysahni.com/best-practices-for-building-a-microservice-architecture

John Kinsella, Cem Gurkok, Frank Geck	Challenges in Securing Application Containers and Microservices	Cloud Security Alliance	2019	Grey	Whitepaper	Google	Included	Excluded	0	Excluded	https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=2&cad=rja&uact=8&ved=2ahUKEwjr
Eric Boersma	Top 10 security traps to avoid when migrating from a monolith to microservices	Sqreen	2019	Grey	Blog post	Google	Included	Included	1	Included	https://blog.sqreen.com/top-10-security-traps-to-avoid-when-migrating-from-a-monolith-to-
Joel Suomalainen	DEFENSE IN DEPTH METHODS IN MICROSERVICES ACCESS CONTROL	Tampere University	2019	Grey	Master's Thesis	Google	Excluded	Excluded	1	Excluded	https://tropo.tuni.fi/bitstream/handle/123456789/27172/suomalainen.pdf?sequence=4&isAllowed=y
Lambda Test	Does Microservices Architecture Influence Security Testing?	Lambda Test	2018	Grey	Blog post	Google	Excluded	Excluded	1	Excluded	https://www.lambdatest.com/blog/does-microservices-architecture-influence-security-testing/
Brian Pitta	What Do Microservices Mean for AppSec?	veracode	2017	Grey	Blog post	Google	Excluded	Excluded	1	Excluded	https://www.veracode.com/blog/managing-appsec/what-do-microservices-mean-
Len Fernandez	42Crunch Announces Full Kubernetes Support to Automate Zero Trust API Security Across Microservices Architecture	prnewswire	2019	Grey	Blog post	Google	Excluded	Excluded	1	Excluded	https://www.prnewswire.com/news-releases/42crunch-announces-full-kubernetes-support-to-automate-zero-
Haim Helman	Scaling Microservices Poses Serious Security Challenges: Haim Helman	TFiR	2019	Grey	Video	Google	Excluded	Included	0	Excluded	https://youtu.be/GAVpE_gQetI
John Carnell	Securing your microservices	Manning	2017	Grey	Book chapter	Google	Included	Included	1	Included	https://livebook.manning.com/book/spring-microservices-in-action/chapter-
WALLARM	A CISO's GUIDE TO CLOUD APPLICATION SECURITY	WALLARM	2019	Grey	Whitepaper	Google	Excluded	Excluded	1	Excluded	https://wallarm.com/files/resources/CISOs%20Guide%20to%20AppSec.pdf
Stephen Doxsee	Implementing Microservices Security Patterns and Protocols with Spring Security	InfoQ	2020	Grey	Video	Google	Excluded	Excluded	1	Excluded	https://www.infoq.com/presentations/microservices-spring-security-5-1/?itm_campaign=rightbar
Ramaswamy Chandramouli	Security Strategies for Microservices-based Application Systems	NIST	2019	Grey	Whitepaper	Google	Included	Included	1	Included	https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-204.pdf

Matt Raible	Security Patterns for Microservice Architectures	Okta	2020	Grey	Blog post	Google	Duplicate	Duplicate	1	Excluded	https://developer.okta.com/blog/2020/03/23/microservice-security-patterns
Edureka	Microservices Security | Best Practices To Secure Microservices | Edureka	edureka!	2019	Grey	Video	Google	Included	Included	1	Included	https://youtu.be/wpA0N7kHaDo
Sam Newman	Security and Microservices by Sam Newman	Devoxx	2016	Grey	Video	Google	Included	Included	1	Included	https://youtu.be/ZXGaC3GR3zU
~~Joe Grandja~~	~~Implementing Microservices Security Patterns and Protocols with Spring Security~~	~~SpringDeveloper~~	~~2019~~	~~Grey~~	~~Video~~	~~Google~~	Excluded	Excluded	1	Excluded	https://youtu.be/JnYIsvJY7gM
~~Adib Saikali~~	~~Microservices Security with Spring~~	~~SpringDeveloper~~	~~2016~~	~~Grey~~	~~Video~~	~~Google~~	Excluded	Excluded	1	Excluded	https://youtu.be/cKjgkNt-tFg
~~Herman Leybovich~~	~~Microservices for Military Applications~~	~~HashiCorp~~	~~2019~~	~~Grey~~	~~Video~~	~~Google~~	Excluded	Excluded	1	Excluded	https://youtu.be/a-Fnnq1h_T8
~~David Blevins~~	~~David Blevins -- Deconstructing REST Security, Iterate 2018~~	~~OktaDev~~	~~2018~~	~~Grey~~	~~Video~~	~~Google~~	Included	Excluded	0	Excluded	https://youtu.be/XuhKdy7UIoY
~~Dallas Monson~~	~~Microservices Anti-Patterns~~	~~Dzone~~	~~2019~~	~~Grey~~	~~Blog post~~	~~Google~~	Excluded	Excluded	1	Excluded	https://dzone.com/articles/microservices-anti-patterns
Michael Hofmann, Erin Schnabel, Katherine	Application security - Microservices Best Practices for Java	IBM Redbooks	2016	Grey	Book chapter	Google	Included	Included	1	Included	https://books.google.cl/books?id=KdSrDQAAQBAJ&pg=PA71&lpg=PA71&dq=microservice+antipattern+s
Vinicius Feitosa Pacheco	Microservice Patterns and Best Practices	Packt	2018	Grey	Book	Google	Included	Included	1	Included	https://books.google.cl/books?id=oyZKDwAAQBAJ&pg=PA142&lpg=PA142&dq=microservice+antipattern
~~NCSC~~	~~Security architecture anti patterns~~	~~NCSC~~	~~2019~~	~~Grey~~	~~Whitepaper~~	~~Google~~	Excluded	Excluded	1	Excluded	https://www.ncsc.gov.uk/whitepaper/security-architecture-anti-patterns

~~Arif Khan~~	~~Microservices: The Good, the Bad, and the Ugly~~	~~Dzone~~	~~2018~~	~~Grey~~	~~Blog post~~	~~Google~~	Excluded	Excluded	1	Excluded	https://dzone.com/articles/microservices-the-good-the-bad-and-the-ugly
Arif Khan	How to Secure Your Microservices — Shopify Case Study	Dzone	2018	Grey	Blog post	Google	Included	Included	1	Included	https://dzone.com/articles/bountytutorial-microservices-security-how-to-secur
Rodrigo Candido da Silva	Best Practices to Protect Your Microservices Architecture	Medium	2017	Grey	Blog post	Google	Included	Included	1	Included	https://medium.com/@rcandidosilva/best-practices-to-protect-your-microservices-architecture-541e7cf7637f
~~Ashan Fernando~~	~~What I have learned Architecting Microservices~~	~~Hackernoon~~	~~2018~~	~~Grey~~	~~Blog post~~	~~Google~~	Excluded	Excluded	1	Excluded	https://hackernoon.com/what-i-have-learned-architecting-microservices-cbccc2182530
Scott Matteson	10 tips for securing microservice architecture	Techrepublic	2017	Grey	Blog post	Google	Included	Included	1	Included	https://www.techrepublic.com/article/10-tips-for-securing-microservice-architecture/
Radware	Microservice Architectures Challenge Traditional Security Practices	Radware	2020	Grey	Blog post	Google	Included	Excluded	0	Included	https://blog.radware.com/security/2020/01/microservice-architectures-challenge-traditional-security-
Joydip Kanjilal	4 fundamental microservices security best practices	SearchAppArchitecture	2020	Grey	Blog post	Google	Included	Included	1	Included	https://searchapparchitecture.techtarget.com/tip/4-fundamental-microservices-security-best-practices
Scott Matteson	How to establish strong microservice security using SSL, TLS and API gateways	Techrepublic	2017	Grey	Blog post	Google	Included	Included	1	Included	https://www.techrepublic.com/article/how-to-establish-strong-microservice-security-using-ssl-tls-and-
~~Warwick Ashford~~	~~Microservices introduce hidden security complexity, analyst warns~~	~~Computer Weekly~~	~~2019~~	~~Grey~~	~~Blog post~~	~~Google~~	Excluded	Excluded	1	Excluded	https://www.computerweekly.com/news/252462690/Microservices-introduce-hidden-security-complexity-
Prabath Siriwardena, Nuwan Dias	Microservices security in action	Manning	2020	Grey	Book	Google	Included	Included	1	Included	https://livebook.manning.com/book/microservices-security-in-action/welcome/v-7/
Matt McLarty, Rob Wilson, and Scott Morrison	Securing Microservice APIs	O'Reilly	2018	Grey	Book	Google	Included	Included	1	Included	https://secureservercdn.net/198.71.233.44/e3z.729.myftpupload.com/wp-content/uploads/2020/01/

Farshad Abasi	Securing modern API- and microservices-based apps by design	IBM	2019	Grey	Blog post	Google	Included	Included	1	Included	https://developer.ibm.com/technologies/api/articles/securing-modern-api-and-microservices-apps-1/
BILL DOERRFELD	How To Control User Identity Within Microservices	Nordic Apis	2018	Grey	Blog post	Google	Excluded	Included	0	Included	https://nordicapis.com/how-to-control-user-identity-within-microservices/
Natasha Gupta	Security Strategies for DevOps, APIs, Containers and Microservices	Imperva	2018	Grey	Blog post	Google	Included	Included	1	Included	https://www.imperva.com/blog/security-strategies-for-devops-apis-containers-and-microservices/
Renata Budko	Five Things You Need to Know About API Security	TheNewStack	2018	Grey	Blog post	Google	Included	Included	1	Included	https://thenewstack.io/5-things-you-need-to-know-about-api-security/
~~Ruth Reinicke~~	~~Authorization and Authentication with Microservices~~	~~LeanIX~~	~~2017~~	~~Grey~~	~~Blog post~~	~~Google~~	Excluded	Excluded	1	Excluded	https://www.leanix.net/en/blog/authorization-authentication-with-microservices
~~Outpost24~~	~~Improve Security of Docker, Containers, and Microservices~~	~~Outpost24~~	~~2017~~	~~Grey~~	~~Blog post~~	~~Google~~	Excluded	Excluded	1	Excluded	https://outpost24.com/blog/Improve-Security-Docker-Containers-Microservices
WALLARM	Shift to Microservices: Evolve Your Security Practices & Container Security	WALLARM	2019	Grey	Blog post	Google	Included	Included	1	Included	https://lab.wallarm.com/shift-to-microservices-evolve-your-security-practices-container-security/
~~Aater Suleman~~	~~Living In A Microservice World: Why Kubernetes Security Matters~~	~~Forbes~~	~~2019~~	~~Grey~~	~~Blog post~~	~~Google~~	Excluded	Excluded	1	Excluded	https://www.forbes.com/sites/forbestechcouncil/2019/02/21/living-in-a-microservice-world-why-
Michael Douglass	Microservices Authentication & Authorization Best Practice	CodeBurst	2018	Grey	Blog post	Google	Included	Included	1	Included	https://codeburst.io/i-believe-it-really-depends-on-your-environment-and-how-well-protected-the-
~~John Au-Yeung~~	~~Best practices for REST API design~~	~~Stack Overflow~~	~~2020~~	~~Grey~~	~~Blog post~~	~~Google~~	Excluded	Excluded	1	Excluded	https://stackoverflow.blog/2020/03/02/best-practices-for-rest-api-design/
Rami Sass	Security in the world of microservices	ITProPortal	2017	Grey	Blog post	Google	Included	Included	1	Included	https://www.itproportal.com/features/security-in-the-world-of-microservices/

~~Sekwon Choi~~	~~How Netflix brings safer and faster streaming experiences to the living room on crowded networks using TLS 1.3~~	~~Netflix Technology Blog~~	~~2020~~	~~Grey~~	~~Blog post~~	~~Google~~	Excluded	Excluded	1	Excluded	https://netflixtechblog.com/how-netflix-brings-safer-and-faster-streaming-experience-to-the-living-
Scott Behrens and Bryan Payne	Starting the Avalanche. Application DDoS In Microservice Architectures	Netflix Technology Blog	2017	Grey	Blog post	Google	Included	Included	1	Included	https://netflixtechblog.com/starting-the-avalanche-640e69b14a06
~~Ryan Bagnulo~~	~~Securing the Microservices Mesh With an API Gateway~~	~~Akana~~	~~2018~~	~~Grey~~	~~Blog post~~	~~Google~~	Excluded	Excluded	1	Excluded	https://www.akana.com/blog/securing-microservices-mesh-api-gateway
~~Darrin Solomon~~	~~Have a safe microservice journey!~~	~~MuleSoft~~	~~2019~~	~~Grey~~	~~Blog post~~	~~Google~~	Excluded	Excluded	1	Excluded	https://blogs.mulesoft.com/dev/microservices-dev/safe-microservice-journey/
~~ThreatAware~~	~~Security by design: how does it work in practice?~~	~~ThreatAware~~	~~2019~~	~~Grey~~	~~Blog post~~	~~Google~~	Excluded	Excluded	1	Excluded	https://threataware.com/security-by-design-how-does-it-work-in-practice/
Scott Behrens, Bryan Payne	Starting the Avalanche: Application DDoS in Microservice Architectures	Netflix Tech Blog	2017	Grey	Blog post	Google	Included	Included	1	Included	https://netflixtechblog.com/starting-the-avalanche-640e69b14a06
VIRAG MODY	From Zero to Zero Trust	Gravitational	2020	Grey	Blog post	Google	Included	Excluded	0	Included	https://gravitational.com/blog/zero-to-zero-trust/
~~Eric Sheridan~~	~~Microservices Security~~	~~WhiteHatSec~~	~~2019~~	~~Grey~~	~~Blog post~~	~~Google~~	Excluded	Excluded	1	Excluded	https://www.whitehatsec.com/blog/microservices-security/
Aaron Parecki	OAuth: When Things Go Wrong	OktaDev	2019	Grey	Video	Google	Included	Included	1	Included	https://www.youtube.com/watch?v=H6MxsFMAoP8
~~Keith Casey, Matt Raible~~	~~Oktane18: API and Microservices Best Practices~~	~~Okta~~	~~2018~~	~~Grey~~	~~Video~~	~~Google~~	Excluded	Excluded	1	Excluded	https://www.youtube.com/watch?v=paEIeZyeJyI
~~Andrew Slivker~~	~~API Security Challenges and How to Address Them~~	~~Nordic APIs~~	~~2018~~	~~Grey~~	~~Video~~	~~Google~~	Excluded	Excluded	1	Excluded	https://www.youtube.com/watch?v=gMEUAwztRMA&feature=emb_logo

Tom Smith	How to Secure APIs	Dzone	2019	Grey	Blog post	Google	Included	Included	1	Included	https://dzone.com/articles/how-to-secure-apis
~~Ben Sigelman~~	~~What We Got Wrong: Lessons from the Birth of Microservices~~	~~InfoQ~~	~~2019~~	~~Grey~~	~~Video~~	~~Google~~	Excluded	Excluded	1	Excluded	https://www.infoq.com/presentations/google-lessons-microservices/
Gerry Gebel and David Brossard	Webinar: Securing APIs and Microservices with OAuth, OpenID Connect, and ABAC	Axiomatics	2018	Grey	Video	Google	Included	Included	1	Included	https://www.youtube.com/watch?v=TnCPJUV9RnA
~~Jean Louis-Monteiro, David Blevins~~	~~Implementing Microservice Security via JWT and MicroProfile~~	~~Oracle Developers~~	~~2019~~	~~Grey~~	~~Video~~	~~Google~~	Excluded	Excluded	1	Excluded	https://youtu.be/_iB9SVjWuo8
~~Phil Wittmer~~	~~The Top Microservices Disadvantages & Advantages~~	~~Tiempo Development~~	~~2019~~	~~Grey~~	~~Blog post~~	~~Google~~	Excluded	Excluded	1	Excluded	https://www.tiempodev.com/blog/disadvantages-of-a-microservices-architecture/
~~Srinath Perera~~	~~Walking the Microservices Path Towards Loose Coupling? Look out for These Pitfalls~~	~~Dzone~~	~~2016~~	~~Grey~~	~~Blog post~~	~~Google~~	Excluded	Excluded	1	Excluded	https://dzone.com/articles/walking-the-microservices-path-towards-loose-coupl
Srinath Perera	Walking the wire: Mastering the Four Decisions in Microservices Architecture	Medium	2016	Grey	Blog post	Google	Excluded	Included	0	Included	https://medium.com/systems-architectures/walking-the-microservices-path-towards-loose-coupling-
~~Philippe De Ryck~~	~~GOTO 2019 • Common API Security Pitfalls • Philippe De Ryck~~	~~GOTO Conferences~~	~~2019~~	~~Grey~~	~~Video~~	~~Google~~	Excluded	Excluded	1	Excluded	https://youtu.be/Ss1tZjooo9I
~~K&C Team~~	~~KUBERNETES AT THE FOREFRONT OF SECURE MICROSERVICES FUTURE~~	~~K&C~~	~~2020~~	~~Grey~~	~~Blog post~~	~~Google~~	Excluded	Excluded	1	Excluded	https://kruschecompany.com/kubernetes-at-the-forefront-of-secure-microservices-future/
~~Ranga Rajagopalan~~	~~Application Services 101 | Dodging Microservices Pitfalls~~	~~AVI Networks~~	~~2016~~	~~Grey~~	~~Blog post~~	~~Google~~	Excluded	Excluded	1	Excluded	https://blog.avinetworks.com/application-services-101
Robert Lemos	App security in the microservices age: 4 best practices	TechBeacon	2019	Grey	Blog post	Google	Included	Included	1	Included	https://techbeacon.com/app-dev-testing/app-security-microservices-age-4-best-practices

Ranny Nachmias	Cloud Native Security Best Practices	Container Journal	2019	Grey	Blog post	Google	Excluded	Included	0	Excluded	https://containerjournal.com/topics/container-security/cloud-native-security-best-practices/
RALPH JANKE	SECURITY CONSIDERATION FOR MICROSERVICES USING CONTAINER TECHNOLOGY	Sector	2017	Grey	Video	Google	Excluded	Excluded	1	Excluded	https://sector.ca/sessions/security-consideration-for-microservices-using-container-technology/
ZACH GARDNER	Security in the Microservices Paradigm	KEYHOLE Software	2017	Grey	Blog post	Google	Duplicate	Duplicate	1	Excluded	https://keyholesoftware.com/2017/03/13/security-in-the-microservices-paradigm/
Cameron Gain	Microservices Security: Probably Not What You Think It Is	TheNewStack	2018	Grey	Blog post	Google	Excluded	Excluded	1	Excluded	https://thenewstack.io/microservices-security-probably-not-what-you-think-it-is/
Tom Smith	How Do You Secure Microservices?	Dzone	2017	Grey	Blog post	Google	Included	Included	1	Included	https://dzone.com/articles/how-do-you-secure-microservices
Umberto Azadi ; Francesca Arcelli	Architectural Smells Detected by Tools: a Catalogue Proposal	International Conference on Technical Debt	2019	White	Conference	IEEE Xplore	Excluded	Excluded	1	Excluded	https://ieeexplore.ieee.org/abstract/document/8785058
Katja Tuma, Danial Hosseini, Kyriakos	Inspection guidelines to identify security design flaws	European Conference on Software Architecture	2019	White	Conference	ACM Digital Library	Excluded	Excluded	1	Excluded	https://dl.acm.org/doi/abs/10.1145/3344948.3344995
J. Bogner, T. Boceck, M. Popp, D. Tschechlov,	Towards a Collaborative Repository for the Documentation of Service-Based Antipatterns and Bad Smells	International Conference on Software Architecture	2019	White	Conference	IEEE Xplore	Excluded	Excluded	1	Excluded	https://ieeexplore.ieee.org/abstract/document/8712355
Andrei Furda Colin Fidge Alistair Barros Olaf	Reengineering Data Centric Information Systems for the Cloud — A Method and Architectural Patterns Promoting Multitenancy	Software Architecture for Big Data and the	2017	White	Book chapter	Scopus	Excluded	Excluded	1	Excluded	http://www.sciencedirect.com/science/article/pii/B9780128054673000132
Ramaswamy Chandramouli	Security Strategies for Microservices based Application Systems	Special Publication (NIST SP)	2019	Grey	Whitepaper	Google Scholar	Excluded	Excluded	1	Excluded	https://www.nist.gov/publications/security-strategies-microservices-based-application-systems
A. Nehme, V. Jesus, K. Mahbub and A. Abdallah	Securing Microservices	IT Professional	2019	White	Journal	IEEE Xplore	Included	Included	1	Included	https://ieeexplore.ieee.org/abstract/document/8657392

T. Yarygina and A. H. Bagge	Overcoming Security Challenges in Microservice Architectures	Symposium on Service-Oriented System	2018	White	Conference	IEEE Xplore	Included	Included	1	Included	https://ieeexplore.ieee.org/abstract/document/8359144
Fangchao Tian ; Peng Liang ; Muhammad	How Developers Discuss Architecture Smells? An Exploratory Study on Stack Overflow	International Conference on Software Architecture	2019	White	Conference	IEEE Xplore	Excluded	Excluded	1	Excluded	https://ieeexplore.ieee.org/abstract/document/8703915
A R Manu ; Jitendra Kumar Patel ; Shakil Akhtar ;	Docker container security via heuristics based multilateral security conceptual and pragmatic study	International Conference on Circuit, Power and	2016	White	Conference	IEEE Xplore	Excluded	Excluded	1	Excluded	https://ieeexplore.ieee.org/abstract/document/7530217
Davide Taibi, Nabil El Ioini, Claus Pahl, Jan Raphael	Serverless Cloud Computing (Function as a Service) Patterns: A Multivocal Literature Review	International conference on Cloud Computing	2020	White	Conference	Scholar	Excluded	Excluded	1	Excluded	https://www.researchgate.net/profile/Davide_Taibi/publication/340121613_Patterns_for_Serverless_Funct
GiulianoCasaleCristinaChestaPeterDeussenElisabettaDi	Current and Future Challenges of Software Engineering for Services and Applications	Procedia Computer Science	2016	White	Journal	Scopus	Excluded	Excluded	1	Excluded	https://www.sciencedirect.com/science/article/pii/S1877050916320944?via%3Dihub
Seyed Yahya Nikouei, Yu Chen, Alexander	I-SAFE: instant suspicious activity identification at the edge using fuzzy decision making	ACM/IEEE Symposium on Edge Computing	2019	White	Conference	ACM Digital Library	Excluded	Excluded	1	Excluded	https://dl.acm.org/doi/abs/10.1145/3318216.3363307
Sahil Suneja, Ali Kanso, and Canturk Isci	Can Container Fusion Be Securely Achieved?	International Workshop on Container Technologies	2019	White	Conference	ACM Digital Library	Excluded	Excluded	1	Excluded	https://dl.acm.org/doi/abs/10.1145/3366615.3368356
Christian Esposito ; Aniello Castiglione ;	Challenges in Delivering Software in the Cloud as Microservices	IEEE Cloud Computing	2016	White	Journal	IEEE Xplore	Included	Included	1	Included	https://ieeexplore.ieee.org/abstract/document/7742281
Mohammad Khodaei, Hamid Noroozi, and	Scaling pseudonymous authentication for large mobile systems	Conference on Security and Privacy in Wireless	2019	White	Conference	ACM Digital Library	Excluded	Excluded	1	Excluded	https://dl.acm.org/doi/abs/10.1145/3317549.3323410
Gastón Márquez and Hernán Astudillo	Identifying availability tactics to support security architectural design of microservice based systems	European Conference on Software Architecture	2019	White	Conference	ACM Digital Library	Excluded	Excluded	1	Excluded	https://dl.acm.org/doi/abs/10.1145/3344948.3344996
Jie Liang ; Mingzhe Wang ; Yuanliang	Fuzz testing in practice: Obstacles and solutions	International Conference on Software Analysis,	2018	White	Conference	IEEE Xplore	Excluded	Excluded	1	Excluded	https://ieeexplore.ieee.org/abstract/document/8330260

Kasun IndrasiriPrabath Siriwardena	Microservices Security Fundamentals	Microservices for the Enterprise. Apress,	2018	White	Book chapter	SpringerLink	Included	Included	1	Included	https://link.springer.com/chapter/10.1007/978-1-4842-3858-5_11
Sourabh Sharma	Securing Microservices	Mastering Microservices with Java - Third Edition -	2019	Grey	Book chapter	Scholar	Included	Included	1	Included	https://www.packtpub.com/application-development/mastering-microservices-java-third-
Nic Jackson	Security	Building Microservices with Go - Packt	2017	Grey	Book chapter	Scholar	Included	Included	1	Included	https://www.packtpub.com/application-development/building-microservices-go
Eberhard Wolff	Security	Microservices: Flexible Software Architecture -	2016	Grey	Book chapter	Scholar	Included	Included	1	Included	https://www.oreilly.com/library/view/microservices-flexible-software/9780134650449/
~~Zhibo Yu ; Jialo Han ; Tianpu Zhao ; Ning Tian ;~~	~~Research and Implementation of Online Judgment System Based on Micro Service~~	~~International Conference on Software Engineering~~	~~2019~~	~~White~~	~~Conference~~	~~IEEE Xplore~~	Excluded	Excluded	1	Excluded	https://ieeexplore.ieee.org/abstract/document/9040684
~~Silvia Esparrachiari Ghirotti, Tanya Reilly,~~	~~Tracking and Controlling Microservice Dependencies~~	~~Commun. ACM 61~~	~~2018~~	~~White~~	~~Magazine~~	~~ACM Digital Library~~	Excluded	Excluded	1	Excluded	https://dl.acm.org/doi/10.1145/3267118
Sam Newman	Security	Building Microservices - O'Reilly Media, Inc.	2015	Grey	Book chapter	Scholar	Included	Included	1	Included	https://www.oreilly.com/library/view/building-microservices/9781491950340/
Justus Bogner ; Jonas Fritzsch ; Stefan	Microservices in Industry: Insights into Technologies, Characteristics, and Software Quality	International Conference on Software Architecture	2019	White	Conference	IEEE Xplore	Excluded	Included	0	Included	https://ieeexplore.ieee.org/abstract/document/8712375
~~S. Sultan, I. Ahmad and T. Dimitriou~~	~~Container Security: Issues, Challenges, and the Road Ahead~~	~~IEEE Access~~	~~2019~~	~~White~~	~~Journal~~	~~IEEE Xplore~~	Excluded	Excluded	1	Excluded	https://ieeexplore.ieee.org/abstract/document/8693491
Tarek Ziadé	Securing Your Services	Python Microservices Development - Packt	2017	Grey	Book chapter	Scholar	Included	Included	1	Included	https://www.packtpub.com/web-development/python-microservices-development
~~Zirak Zaheer, Hyunseok Chang, Sarit Mukherjee,~~	~~EZTrust: Network-Independent Zero-Trust Perimeterization for Microservices~~	~~ACM Symposium on SDN Research~~	~~2019~~	~~White~~	~~Conference~~	~~ACM Digital Library~~	Excluded	Included	0	Excluded	https://dl.acm.org/doi/abs/10.1145/3314148.3314349

Richard Takashi Freeman	Securing Your Microservices	Building Serverless Microservices in Python —	2019	Grey	Book chapter	Scholar	Excluded	Excluded	1	Excluded	https://www.packtpub.com/application-development/building-serverless-microservices-
Vlad Bucur ; Ovidiu Stan ; Liviu C. Miclea	Data Loss Prevention and Data Protection in Cloud Environments Based on Authentication Tokens	International Conference on Control Systems and	2019	White	Conference	IEEE Xplore	Excluded	Excluded	1	Excluded	https://ieeexplore.ieee.org/abstract/document/8744776
T. Salman, M. Zolanvari, A. Erbad, R. Jain and M.	Security Services Using Blockchains: A State of the Art Survey	IEEE Communications Surveys & Tutorials	2019	White	Journal	IEEE Xplore	Excluded	Excluded	1	Excluded	https://ieeexplore.ieee.org/abstract/document/8428402
Madiha H. Syed and Eduardo B. Fernandez	The secure container manager pattern	Conference on Pattern Languages of Programs	2018	White	Conference	ACM Digital Library	Excluded	Excluded	1	Excluded	https://dl.acm.org/doi/abs/10.5555/3373669.3373676
Graham Lea	Microservices Security: All The Questions You Should Be Asking	GrahamLea	2015	Grey	Blog post	Google	Included	Included	1	Included	https://www.grahamlea.com/2015/07/microservices-security-questions/
Yuqiong Sun ; Susanta Nanda ; Trent Jaeger	Security as a Service for Microservices Based Cloud Applications	International Conference on Cloud Computing	2015	White	Conference	IEEE Xplore	Included	Excluded	0	Excluded	https://ieeexplore.ieee.org/abstract/document/7396137
Daniel Richter, Tim Neumann and Andreas Polze	Security Considerations for Microservice Architectures	International Conference on Cloud Computing	2018	White	Conference	Scholar	Included	Included	1	Included	https://www.scitepress.org/Papers/2018/67910/67910.pdf
Christopher Gerking ; David Schubert	Component Based Refinement and Verification of Information Flow Security Policies for Cyber Physical Microservice Architectures	International Conference on Software Architecture	2019	White	Conference	IEEE Xplore	Excluded	Excluded	1	Excluded	https://ieeexplore.ieee.org/abstract/document/8703909
Chien An Chen	With Great Abstraction Comes Great Responsibility: Scaling the Microservices Attack Surface	IEEE Cybersecurity Development (SecDev)	2019	White	Conference	IEEE Xplore	Excluded	Excluded	1	Excluded	https://ieeexplore.ieee.org/abstract/document/8901600
Reza Tourani, Austin Bos, Satyajayant Misra, and	Towards security as a service in multi access edge	ACM/IEEE Symposium on Edge Computing	2019	White	Conference	ACM Digital Library	Excluded	Excluded	1	Excluded	https://dl.acm.org/doi/abs/10.1145/3318216.3363335
Z. Wen et al	GA Par: Dependable Microservice Orchestration Framework for Geo Distributed Clouds	IEEE Transactions on Parallel and	2020	White	Journal	IEEE Xplore	Excluded	Excluded	1	Excluded	https://ieeexplore.ieee.org/abstract/document/8766876

Jiantao Zhao and Jin Sun	Research on Access Control Model Based on RBAC Model in Microservice Environment	International Symposium on Big Data and Applied	2019	White	Conference	Scopus	Excluded	Excluded	1	Excluded	https://iopscience.iop.org/article/10.1088/1742-6596/1437/1/012031/meta
Peter Nkomo, Marijke Coetzee	Software Development Activities for Secure Microservices	International Conference on Computation	2019	White	Conference	SpringerLink	Included	Included	1	Included	https://link.springer.com/chapter/10.1007/978-3-030-24308-1_46
Antonio Nehme, Vitor Jesus, Khaled Mahbub, Ali	Fine-Grained Access Control for Microservices	International Symposium on Foundations	2018	White	Conference	SpringerLink	Included	Included	1	Included	https://link.springer.com/chapter/10.1007/978-3-030-18419-3_19
Kennedy A. Torkura ; Muhammad I.H. Sukmana	A Cyber Risk Based Moving Target Defense Mechanism for Microservice Architectures	Intl Conf on Parallel & Distributed Processing	2018	White	Conference	IEEE Xplore	Excluded	Excluded	1	Excluded	https://ieeexplore.ieee.org/abstract/document/8672278/
M. Özbek and M. T. Sandıkkaya	Detecting Malicious Behavior in Microservice Based Web Applications	Signal Processing and Communicati	2019	White	Conference	IEEE Xplore	Excluded	Excluded	1	Excluded	https://ieeexplore.ieee.org/abstract/document/8806294/
A. Pereira Valo, G. Marquez, H. Astudillo, E.	Security Mechanisms Used in Microservices Based Systems: A Systematic Mapping	2019 XLV Latin American Computing	2019	White	Conference	IEEE Xplore	Excluded	Included	0	Excluded	https://ieeexplore.ieee.org/abstract/document/9073967/authors
Leon O'Neill	MICROSERVICE SECURITY – WHAT YOU NEED TO KNOW	Crashtest Security	2020	Grey	Blog post	Google	Included	Included	1	Included	https://crashtest-security.com/microservice-security-what-you-need-to-know/
Matt Raible	11 Patterns to Secure Microservice Architectures	DZone	2020	Grey	Blog post	Google	Included	Included	1	Included	https://dzone.com/articles/11-patterns-to-secure-microservice-architectures?edition=613
Md Kamaruzzaman	Microservice Architecture and its 10 Most Important Design Patterns	Towards data science	2020	Grey	Blog post	Google	Included	Included	1	Included	https://towardsdatascience.com/microservice-architecture-and-its-10-most-important-design-
Prabath Siriwardena	Challenges of Securing Microservices	Medium	2020	Grey	Blog post	Google	Included	Included	1	Included	https://medium.facilelogin.com/challenges-of-securing-microservices-68b55877d154
Chandra Rajasekharaiah	Securing Microservices on Cloud	Cloud-Based Microservices. Apress, Berkeley, CA	2020	White	Book chapter	SpringerLink	Included	Included	1	Included	https://link.springer.com/book/10.1007%2F978-1-4842-6564-2

Nuno Mateus-Coelho, Manuela Cruz-Cunha, Luis	Security in Microservices Architectures	International Conference on ENTERprise	2020	White	Conference	Scholar	Included	Included	1	Included	https://www.researchgate.net/profile/Nuno_Mateus-Coelho/publication/329952695_Security_in_Microse

IR: 88.24

Selected Literature

ID	Authors	Title	Venue	Year	Type	Sub-Type	Search engine	Link
1	Marco Troisi	8 best practices for microservices app sec	TechBeacon	2017	Grey	Blog post	Google	https://techbeacon.com/app-dev-testing/8-best-practices-
2	Jack Mannino	Security In The Land Of Microservices	AppSec EU 2017	2017	Grey	Video	Google	https://www.youtube.com/watch?v=JRmWlLY8MGE
3	Zach Gardner	Security in the Microservices Paradigm	Dzone	2017	Grey	Blog post	Google	https://dzone.com/articles/security-in-the-microservices-
4	Sumo Logic	Improving Security in Your Microservices Architecture	Sumo Logic	2019	Grey	Blog post	Google	https://www.sumologic.com/insight/microservices-architecture-
5	Thribhuvan Krishnamurthy	Transition to Microservice Architecture - Challenges	Beingtechie	2018	Grey	Blog post	Google	https://www.beingtechie.io/blog/transition-to-microservices-
6	Vinay Sahni	Best Practices for Building a Microservice Architecture	Vinay Sahni	2019	Grey	Blog post	Google	https://www.vinaysahni.com/best-practices-for-building-a-
7	Eric Boersma	Top 10 security traps to avoid when migrating from a monolith to microservices	Sqreen	2019	Grey	Blog post	Google	https://blog.sqreen.com/top-10-security-traps-to-avoid-when-
8	John Carnell	Securing your microservices	Manning	2017	Grey	Book chapter	Google	https://livebook.manning.com/book/spring-microservices-in-
10	Ramaswamy Chandramouli	Security Strategies for Microservices-based Application Systems	NIST	2019	Grey	Whitepaper	Google	https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.
12	Edureka	Microservices Security | Best Practices To Secure Microservices | Edureka	edureka!	2019	Grey	Video	Google	https://youtu.be/wpA0N7kHaDo
13	Sam Newman	Security and Microservices by Sam Newman	Devoxx	2016	Grey	Video	Google	https://youtu.be/ZXGaC3GR3zU
15	Chintan Jain	Top 10 Security Best Practices to secure your Microservices - AppSecUSA 2017	OWASP	2018	Grey	Video	Google	https://youtu.be/VtUQINsYXDM

16	Prabath Siriwardena	Microservices Security Landscape, WSO2 SF Summit 2019	WSO2	2019	Grey	Video	Google	https://youtu.be/6jGePTpbgtI
17	Michael Hofmann, Erin Schnabel, Katherine Stanley	Application security - Microservices Best Practices for Java	IBM Redbooks	2016	Grey	Book chapter	Google	https://books.google.cl/books?id=KdSrDQAAQBAJ&pg=PA71
18	Vinicius Feitosa Pacheco	Security - Microservice Patterns and Best Practices	Packt	2018	Grey	Book chapter	Google	https://books.google.cl/books?id=oyZKDwAAQBAJ&pg=PA142
19	Arif Khan	How to Secure Your Microservices — Shopify Case Study	Dzone	2018	Grey	Blog post	Google	https://dzone.com/articles/bountytutorial-microservices-
20	Rodrigo Candido da Silva	Best Practices to Protect Your Microservices Architecture	Medium	2017	Grey	Blog post	Google	https://medium.com/@rcandidosilva/best-practices-to-protect-
21	Scott Matteson	10 tips for securing microservice architecture	Techrepublic	2017	Grey	Blog post	Google	https://www.techrepublic.com/article/10-tips-for-securing-
22	Radware	Microservice Architectures Challenge Traditional Security Practices	Radware	2020	Grey	Blog post	Google	https://blog.radware.com/security/2020/01/microservice-
23	Joydip Kanjilal	4 fundamental microservices security best practices	SearchAppArchitecture	2020	Grey	Blog post	Google	https://searchapparchitecture.techtarget.com/tip/4-fundamental-
24	Scott Matteson	How to establish strong microservice security using SSL, TLS and API gateways	Techrepublic	2017	Grey	Blog post	Google	https://www.techrepublic.com/article/how-to-establish-strong-
25	Prabath Siriwardena, Nuwan Dias	Microservices security in action	Manning	2020	Grey	Book	Google	https://livebook.manning.com/book/microservices-security-in-
26	Matt McLarty, Rob Wilson, and Scott Morrison	Securing Microservice APIs	O'Reilly	2018	Grey	Book	Google	https://secureservercdn.net/198.71.233.44/e3z.729.myftpuplo
27	Farshad Abasi	Securing modern API- and microservices-based apps by design	IBM	2019	Grey	Blog post	Google	https://developer.ibm.com/technologies/api/articles/securing-
28	BILL DOERRFELD	How To Control User Identity Within Microservices	Nordic Apis	2018	Grey	Blog post	Google	https://nordicapis.com/how-to-control-user-identity-within-

29	Natasha Gupta	Security Strategies for DevOps, APIs, Containers and Microservices	Imperva	2018	Grey	Blog post	Google	https://www.imperva.com/blog/security-strategies-for-devops-
30	Renata Budko	Five Things You Need to Know About API Security	TheNewStack	2018	Grey	Blog post	Google	https://thenewstack.io/5-things-you-need-to-know-about-api-
31	WALLARM	Shift to Microservices: Evolve Your Security Practices & Container Security	WALLARM	2019	Grey	Blog post	Google	https://lab.wallarm.com/shift-to-microservices-evolve-
32	Michael Douglass	Microservices Authentication & Authorization Best Practice	CodeBurst	2018	Grey	Blog post	Google	https://codeburst.io/i-believe-it-really-depends-on-your-
33	Rami Sass	Security in the World of Microservices	ITProPortal	2017	Grey	Blog post	Google	https://www.itproportal.com/features/security-in-the-world-of-
34	Scott Behrens, Bryan Payne	Starting the Avalanche: Application DDoS in Microservice Architectures	Netflix Tech Blog	2017	Grey	Blog post	Google	https://netflixtechblog.com/starting-the-avalanche-
35	VIRAG MODY	From Zero to Zero Trust	Gravitational	2020	Grey	Blog post	Google	https://gravitational.com/blog/zero-to-zero-trust/
36	Aaron Parecki	OAuth: When Things Go Wrong	OktaDev	2019	Grey	Video	Google	https://www.youtube.com/watch?v=H6MxsFMAoP8
37	Tom Smith	How to Secure APIs	Dzone	2019	Grey	Blog post	Google	https://dzone.com/articles/how-to-secure-apis
38	Gerry Gebel and David Brossard	Webinar: Securing APIs and Microservices with OAuth, OpenID Connect, and ABAC	Axiomatics	2018	Grey	Video	Google	https://www.youtube.com/watch?v=TnCPJUV9RnA
39	Srinath Perera	Walking the wire: Mastering the Four Decisions in Microservices Architecture	Medium	2016	Grey	Blog post	Google	https://medium.com/systems-architectures/walking-
40	Robert Lemos	App security in the microservices age: 4 best practices	TechBeacon	2019	Grey	Blog post	Google	https://techbeacon.com/app-dev-testing/app-security-
42	Tom Smith	How Do You Secure Microservices?	Dzone	2017	Grey	Blog post	Google	https://dzone.com/articles/how-do-you-secure-microservices

43	A. Nehme, V. Jesus, K. Mahbub and A. Abdallah	Securing Microservices	IT Professional	2019	White	Journal	IEEE Xplore	https://ieeexplore.ieee.org/abstract/document/8657392
44	T. Yarygina and A. H. Bagge	Overcoming Security Challenges in Microservice Architectures	Symposium on Service-Oriented System Engineering	2018	White	Conference	IEEE Xplore	https://ieeexplore.ieee.org/abstract/document/8359144
45	Christian Esposito ; Aniello Castiglione ; Kim-Kwang Raymond	Challenges in Delivering Software in the Cloud as Microservices	IEEE Cloud Computing	2016	White	Journal	IEEE Xplore	https://ieeexplore.ieee.org/abstract/document/7742281
46	Kasun IndrasiriPrabath Siriwardena	Microservices Security Fundamentals	Microservices for the Enterprise. Apress, Berkeley, CA	2018	White	Book chapter	SpringerLink	https://link.springer.com/chapter/10.1007/978-1-4842-3858-
47	Sourabh Sharma	Securing Microservices	Mastering Microservices with Java - Third Edition -	2019	Grey	Book chapter	Scholar	https://www.packtpub.com/application-development/masteri
48	Nic Jackson	Security	Building Microservices with Go - Packt	2017	Grey	Book chapter	Scholar	https://www.packtpub.com/application-development/buildin
49	Eberhard Wolff	Security	Microservices: Flexible Software Architecture -	2016	Grey	Book chapter	Scholar	https://www.oreilly.com/library/view/microservices-flexible-
50	Sam Newman	Security	Building Microservices - O'Reilly Media, Inc.	2015	Grey	Book chapter	Scholar	https://www.oreilly.com/library/view/building-
51	Justus Bogner ; Jonas Fritzsch ; Stefan Wagner ; Alfred	Microservices in Industry: Insights into Technologies, Characteristics, and Software Quality	International Conference on Software	2019	White	Conference	IEEE Xplore	https://ieeexplore.ieee.org/abstract/document/8712375
52	Tarek Ziadé	Securing Your Services	Python Microservices Development - Packt	2017	Grey	Book chapter	Scholar	https://www.packtpub.com/web-development/python-
53	Graham Lea	Microservices Security: All The Questions You Should Be Asking	GrahamLea	2015	Grey	Blog post	Google	https://www.grahamlea.com/2015/07/microservices-security-
54	Daniel Richter, Tim Neumann and Andreas Polze	Security Considerations for Microservice Architectures	International Conference on Cloud Computing and	2018	White	Conference	Scholar	https://www.scitepress.org/Papers/2018/67910/67910.pdf
55	Peter Nkomo, Marijke Coetzee	Software Development Activities for Secure Microservices	International Conference on Computational	2019	White	Conference	SpringerLink	https://link.springer.com/chapter/10.1007/978-3-030-24308-

56	Antonio Nehme, Vitor Jesus, Khaled Mahbub, Ali Abdallah	Fine-Grained Access Control for Microservices	International Symposium on Foundations and	2018	White	Conference	SpringerLink	https://link.springer.com/chapter/10.1007/978-3-030-18419-
58	Leon O'Neill	MICROSERVICE SECURITY – WHAT YOU NEED TO KNOW	Crashtest Security	2020	Grey	Blog post	Google	https://crashtest-security.com/microservice-security-what-
59	Matt Raible	11 Patterns to Secure Microservice Architectures	DZone	2020	Grey	Blog post	Google	https://dzone.com/articles/11-patterns-to-secure-microservice-
60	Md Kamaruzzaman	Microservice Architecture and its 10 Most Important Design Patterns	Towards data science	2020	Grey	Blog post	Google	https://towardsdatascience.com/microservice-architecture-and-
61	Prabath Siriwardena	Challenges of Securing Microservices	Medium	2020	Grey	Blog post	Google	https://medium.facilelogin.com/challenges-of-securing-
62	Chandra Rajasekharaiah	Securing Microservices on Cloud	Cloud-Based Microservices. Apress, Berkeley, CA	2020	White	Book chapter	SpringerLink	https://link.springer.com/book/10.1007%2F978-1-4842-6564-2
63	Nuno Mateus-Coelho, Manuela Cruz-Cunha, Luis Gonzaga Ferreira	Security in Microservices Architectures	International Conference on ENTERprise	2020	White	Conference	Scholar	https://www.researchgate.net/profile/Nuno_Mateus-

Total 58

Literature Analysis

ID	Smell	Why	Refactoring	Why	Francisco	Jacopo	Agreement
1	**Non-proper access control**	The overwhelming majority of applications are going to need to perform some level of access control and authorization handling. What you want to avoid here is reinventing the wheel.	**Use OAuth 2.0**	OAuth/OAuth2 is practically the industry standard as far as user authorization goes. […] The advantage of using it is that you can rely on libraries and platforms	X	X	1
	No layered security	Assuming that a firewall on your network perimeter is enough to protect your software is a big mistake	**Use defense-in-depth**	"Defense in depth" is defined as "an information assurance concept in which multiple layers of security controls (defense) are placed throughout an information technology system."	X	X	1
	Own crypto code	Don't write your own crypto code	**Use already validated solutions**	the open source tools already available (tools that have been heavily battle tested by the community)	X	X	1
	Publicly accessible microservices	Get your containers out of the public network	**Use firewall for the API gateway**	By using this technique you can secure all of your microservices behind a firewall, allowing the API gateway to handle external requests and then talk to the microservices behind the firewall	X	X	1
2	**Publicly accessible microservices**		**Secure APIs**	Your APIs are the gateway into the microservice architecture	X	X	1
	Insecure infrastructure	With Infrastructure-as-Code, your architecture might be in a GitHub Repo	**Restrict commits**	Important to restrict who can commit to master	X	X	1
	Insecure infrastructure	With Infrastructure-as-Code, your architecture might be in a GitHub Repo	**Review code merges**	Important to review code merges (pull requests, etc)	X	X	1
	Decentralized Authentication		**Use API Gateway**	API Gateway is the most prolific Microservice authentication pattern. Each request is signed, which provides an additional layer of authentication.	X	X	1
	Centralized authorization		**Use a token-based approach**	We have decentralized everything. API Gateway + JWT can help to share information across services.	X	X	1

	Centralized authorization		**Use CQRS**	Command and query interfaces are separated. We have a lot more granular control over which services and users we authorise around capabilities	X	X	1
	Non-standard identity propagation		**Use JSON Web Tokens (JWT)**	JWT allows us to pass identity and claims across services.		X	0
	Hardcoded secrets	Bad ideas: Hardcoding credentials in your code, Hardcoding credentials in your Dockerfile, Using environment variables to pass secrets	**Encrypt secrets at rest**			X	0
3	**Decentralized Authentication**		**Use API Gateway**	the API Gateway is the best place to validate Authentication as well as enforce Authorization. This allows you to ensure that anything that goes from the edge into the ecosystem is correctly Authenticated	X	X	1
	Own crypto code	minimize the amount of code your team writes	**Use already validated solutions**	maximize the amount of code you can leverage from libraries that are bullet proof	X	X	1
4	**Centralized authorization**	monolithic applications. For the latter, security teams often use centralized security modules, which cover authentication, authorization, and a range of other critical security measures. However, in a microservices architecture, this kind of centralization would diminish the advantages of distributed deployment and reduce efficiency	**Use a token-based approach**		X	X	1
	Non-standard Authorization	Most applications within a microservices architecture require methods for controlling access and authorization. [..] However, many security analysts advise against starting from scratch	**Use OAuth 2.0**	recommend using OAuth and OAuth2 for authorization management	X	X	1

	No layered security	A standard firewall on the network perimeter is insufficient for protecting an organization's software architecture	**Use defense-in-depth**	microservices calls for a more robust defense involving multiple layers of security controls, placed throughout the information technology system	X	X	1
	Publicly accessible microservices	While it is possible, in theory, to structure the application to make direct calls to each service, this can lead to highly complex code involving an overwhelming number of service calls	**Use firewall for the API gateway**	By placing the API gateway behind a firewall, you can essentially place a firewall around all of your microservices.	X	X	1
5	**Publicly accessible microservices**		**Use firewall for the API gateway**	API Gateway pattern that provides abstraction, security, auditing and monitoring of the calls from external clients to the underlying microservices	X	X	1
	Non-scalable security controls		**DevSecOps**	DevSecOps approach that helps in integrating security processes, principles, best practices and tools early in the development lifecycle, thereby encouraging collaboration among Security experts and Business Analysts, Architects, Development and Operations team, thus making everyone accountable and responsible for building secured systems.	X	X	1
	Non-scalable security controls		**Continuous Security Testing**	Automating security tests for both application and infrastructure layers and integrating with CI/CD pipeline provides a platform for continuous testing of security elements of the distributed system that you are building.	X	X	1
	Non-standard Authorization		**Use OAuth 2.0**	Authorized access to protected resources at an application level. Consider using OAuth delegated authorization schemes such as JWT	X	X	1

	Non-encrypting sensitive data		**Encrypt data at rest**	try and test encryption algorithms to encrypt data in transit and at rest	X		0
	No layered security		**Use defense-in-depth**	Securing infrastructure-level components such as Containers, SSL communication layer, Firewall, etc.	X	X	1
6	**No layered security**	Rather than assuming a network perimeter firewall is good enough, continue to add multiple layers of security where it matters most	**Use defense-in-depth**	This adds redundancy to your security and also helps slow down an attacker when one layer of security fails or a vulnerability is identified.	X	X	1
	No layered security	Microservices are still subject to the same security vulnerabilities and hardening precautions that more full-featured services are. Servers running microservices are often less robust than monolithic servers, which can make them more susceptible to DDoS attacks.	**Use defense-in-depth**	You can take a major step toward defense-in-depth by moving many of your microservices off of public networks.	X	X	1
	Non-secure communication	anyone who can set up a listener on your service network can view all the information passing between services. Often times, that simply means that they can read all the information present in your system, even very sensitive information.	**Use Transport Layer Security (TLS)**	It's better to be proactive and add another layer of defense here rather than reacting after a breach. Adding TLS to a microservice is cheap and easy. It's not worth the increased risk of leaving it off.	X	X	1
	Non-encrypting sensitive data	Whether that's your database or your file storage, it shouldn't be possible for anyone who can access the hard drive to read your critical data without also having your security keys.	**Encrypt data at rest**	Encrypting data at rest is a best practice for any security context. Microservices don't change that at all, but many teams neglect to do it.	X	X	1

	unnecessary privileges to microservices	Another common pitfall when migrating to microservices is to treat all of them the same. We've touched on this previously, but the truth is that all microservices aren't equal. Some need far more access to critical infrastructure—like databases—than others. Instead of simply allowing all services the same level of access to all architecture, intelligently limit their access. Not every service needs to be able to talk to your database or your file persistence layer.	**Least privilege principle**	By limiting what services can access, you simplify the attack surface if one service were to be compromised.	X		0
	Non-proactive security	Most of the time, we don't "just know" that something's wrong. We might figure it out eventually, but that takes time. The time that passes between something going wrong and finding out that something is wrong increases your risk profile significantly.	**Use attack mitigation techniques**	An Application Security Management solution gives you a heads up when something's going wrong with a service.	X	X	1
8	**Non-standard Authorization**	You need to ensure that the proper user controls are in place so that you can validate that a user is who they say they are and that they have permission to do what they're trying to do.	**Use OAuth 2.0**	OAuth2 is a token-based security framework. The main goal behind OAuth2 is that when multiple services are called to fulfill a user's request, the user can be authenticated by each service without having to present their credentials to each service processing their request.	X	X	1
	Non-secure communication		**Use transport Layer Security (TLS)**	your microservices should communicate only through the encrypted channels provided through HTTPS and SSL. The configuration and setup of the HTTPS can be automated through your DevOps scripts.	X	X	1

Decentralized security policy management	The individual servers, service endpoints, and ports your services are running on should never be directly accessible to the client	**Use API Gateway**	Instead, use a services gateway to act as an entry point and gatekeeper for your service calls. Configure the network layer on the operating system or container your microservices are running in to only accept traffic from the services gateway. Remember, the services gateway can act as a policy enforcement point (PEP) that can be enforced against all services. A service gateway also allows you to lock down what port and endpoints you're going to expose to the outside world.	X	X	1
unnecessary privileges to microservices	Least privilege is the concept that a user should have the bare minimum network access and privileges to do their day-to-day job. To this end, you should implement least-privilege by separating your services into two distinct zones: public and private.	**Least privilege principle**	The public zone contains the public APIs that will be consumed by clients. Public microservices should be behind their own services gateway and have their own authentication service for performing OAuth2 authentication. The private zone acts as a wall to protect your core application functionality and data. The private zone should only be accessible through a single well-known port and should be locked down to only accept network traffic from the network subnet that the private services are running. The private zone should have its own services gateway and authentication service.	X	X	1
unnecessary privileges to microservices		**Lock down unneded network ports**	Limit the attack surface of your microservices by locking down un-nneeded network ports	X		0

10	**Non-standard authentication**	Authentication to microservices APIs that have access to sensitive data should not be done simply by using API keys.	**Use OpenID Connect**	Access to such APIs should require authentication tokens that have either been digitally signed (e.g., client credentials grant) or is verified with an authoritative source. Additionally, some services may require either single-use tokens or short-lived tokens (tokens that expire after a short time period) to limit the damage a compromised token can cause.	X	X	1
	Decentralized Authentication	Unlike a monolithic application where the endpoint may be a single server, a microservices-based application consists of multiple fine-grained endpoints.	**Use API Gateway**	Makes sense to provide a single entry point for all clients to multiple component microservices of the application. An API gateway can simplify the situation for clients by exposing an endpoint that will automatically make all the needed multiple requests (calls) and return a single, aggregated response to the client.		X	0
	Giant API Gateway	To prevent the gateway from having too much logic to handle request shaping for different client types, it is divided into multiple gateways.	**Use Backend for Frontend**	In BFF, each client type is given its own gateway (e.g., web app BFF, mobile app BFF) as a collection point for service requests. The respective backend is closely aligned with the corresponding front end (client) and is typically developed by the same team.	X	X	1

	Non-secure communication	Secure communication between clients and services (north-south traffic) and between services (east-west traffic) is critical for the operation of a microservices-based application.	**Transport Layer Security (TLS) mutual authentication**	Communication between an application service and a service registry should occur through a secure communication protocol such as HTTPS or Transport Layer Security. Client to API gateway as well as Service to Service communication should take place after mutual authentication and be encrypted (e.g., using mutual TLS (mTLS) protocol). Frequently interacting services should create keep-alive TLS connections.	X	X	1
	Non-proactive security	Though it is impossible to protect against all types of Internet-based attacks including botnets, microservice APIs must be provided with detection and prevention capabilities against credential-stuffing and credential abuse attacks as well as the capability to detect malicious botnet.	**Use attack mitigation techniques**	A run-time prevention strategy for credential abuse is preferable to an offline strategy. A threshold for a designated time interval from a given location (e.g., IP address) for the number of login attempts should be established; if the threshold is exceeded, preventive measures must be triggered by the authentication/authorization server.	X	X	1
12	**No layered security**		**Use defense-in-depth**	A technique through which you can apply layers of security countermeasures to protect the sensitive services. Potential attackers cannot crack the security on a single go.	X	X	1
	Decentralized Authentication		**Use API Gateway**	Add an extra element to secure services through token authentication. Acts as an entry point to all the client requests and efficiently hides the microservices from the client.	X	X	1

	Non-standard Authorization	Client details and permissions need to be verified as and when a requests is sent.	**Use OAuth 2.0**	Tokens need to be encrypted to avoid any exploitation from 3rd party resources. While mentioning the access token the client communicates with the authorization server, and this server authorizes the client to prevent others from forging the client's identity.	X	X	1
	Non-secure communication	Applications often face traffic from users, 3rd parties, and microservices communicates with each other. But, since these services are accessed by the 3rd parties, there is always a risk of attacks.	**Transport Layer Security (TLS) mutual authentication**	With mutual SSL, the data transferred between the services will be encrypted.	X	X	1
13	**Non-secure communication**	Those in process calls are now inter-process calls.	**Use Transport Layer Security (TLS)**	HTTPS is a good default choice.	X	X	1
	Non-standard Authorization	Confused Deputy Problem! You can trick an intermediate party into asking for things they shouldn't be able to get.	**Use OAuth 2.0**	Use OAuth to manage authentication and authorization.	X	X	1
	Non-encrypting sensitive data		**Encrypt data at rest**	Identify the most critical databases and encrypt them. Microservices architecture allows focusing your energy and resources.	X	X	1
15	**Non-secure communication**		**Use Transport Layer Security (TLS)**	Encrypt all traffic to your end-point to prevent man in the middle attacks.	X	X	1
	unnecessary privileges to microservices		**Least privilege principle**	Each component in your system architecture should have access control! You need to do proper Authorization at every layer.	X	X	1
	Non-proactive security	Never assume that your microservices can scale infinitely.	**Use attack mitigation techniques**	Rate limit and throttle your microservices traffic. Rate limit protects you against DDoS & availability issues.	X	X	1
	No layered security	You need to design a secure network architecture.	**Use defense-in-depth**	WAF + API Gateway + TLS + Network segmentation	X	X	1

	Hardcoded secrets	Do not store private keys & system credentials in plain text on your servers.	**Encrypt secrets at rest**	Exposure of you private key will negate all your other security controls.	X	X	1
	Non-encrypting sensitive data		**Encrypt data at rest**	You need to encrypt or Hash sensitive data.	X	X	1
16	**Publicly accessible microservices**	You have multiple endpoints that transform into entry points, and therefore the attack surface of your system expands.	**Use API Gateway**	You selectively pick a set of microservices that you want to expose to the outside world and they need yo be expose through an API Gateway.	X	X	1
	Trust the Network	Today we consider it an antipattern (Network trust). Trust nothing but verify everything	**Zero Trust Network**	You need to go for a zero-trust network model. You need to keep policy enforcement point or secret checkpoints as much close to the resource.	X	X	1
	Non-secure communication		**Transport Layer Security (TLS) mutual authentication**	Encrypt all traffic to your endpoints and service to service communication.	X	X	1
17	**No layered security**	Securing the perimeter is not going to be enough. The question is how to secure a rapidly changing infrastructure in an organic way that still allows the individual services to change and grow without requiring central coordination.	**Use defense-in-depth**	Using an extra firewall or gateway to guard resources that require more levels of protection is a good idea. Defined application boundaries provide a reasonable amount of isolation between independently varying systems, and are a good way to maintain a reasonable security posture in a dynamic enviroment.	X	X	1
	Non-encrypting sensitive data		**Encrypt data at rest**	Sensitive data should be encrypted as early as possible, and decrypted as late as possible. If sensitive data must flow between services, only do so while it is encrypted, and should not be decrypted until the data needs to be used. This process can help prevent accidental exposure in logs, for example.	X	X	1

Decentralized Authentication		**Use API Gateway**	It is common to have authentication (establishing the user's identity) performed by a dedicated, centralized service or evan an API gateway.	X		0
Centralized authorization	With a monolithic application, it is common to have fine grained roles, or at least role associated groups in a central user repository. With the emphasis on independent lifecycles of microservices, however, this dependency is an anti-pattern. Development of an independent microservice is then constrained by and coupled with updates to the centralized resource.	**Decentralize authorization**	When working with authorisation (establishing a user's authority or permission to access a secured resource), in a microservice environment keep group or role definiitions coarse grained in common, cross cutting services. Allow individual services to maintain their own fine grained controls.	X		0
Own crypto code		**Use already validated solutions**	Use known data encryption technologies rather than inventing your own	X	X	1
Hardcoded secrets	Secrets should never be hardcoded because that increases the likelihood they will be compromised. It either makes your code sensitive to the enviroment, or requires different environments to use the same shared secret, neither of which are good practices.	**Encrypt secrets at rest**	Do not store your credentials alongside your applications. Do not store your credentials in a public repository. only store encrypted values.	X	X	1
Non-scalable security controls		**DevSecOps**	Repeatable automated processes should be used for applying security policies, credentials, and managing SSL certificates and keys across segmented enviroments to help avoid human error.	X	X	1

	Trust the Network	It is inherently unsafe to assume a secure private network. End-to-end SSL can bring some benefit in that bytes are not flowing around in plain text, but it does not establish a trusted enviroment on its own, and requires key management.	**Zero Trust Network**		X	X	1
	Non-standard Authorization	You must establish and maintain the identity of users without introducing extra latency and contention with frequent calls to a centralized service.	**Use OAuth 2.0**	OAuth provides an open framework for delegating authorization to a third party.	X	X	1
	Non-standard authentication	You must establish and maintain the identity of users without introducing extra latency and contention with frequent calls to a centralized service.	**Use OpenID Connect**	OIDC provides an identity layer on top of OAuth 2.0, making it possible to also delegate authentication to a third party.	X	X	1
	Non-standard identity propagation	Identity propagation is another challenge in microservices enviroment. After the user has been authenticated, that identity needs to be propagated to the next service in a trusted way.	**Use JSON Web Tokens (JWT)**	JWTs can be used to carry along a representation of information about the user. In essence, you want to be able to accomplish these tasks: Know that the request was initiated from a user request. Know the identity that the request was made on behalf of. Know that this request is not a malicious replay of a previous request.	X	X	1
18	**Non-secure communication**	When we are working with APIs, we need to think about the security of data traffic and especially the level of permission that each user should have.	**Use JSON Web Tokens (JWT)**	There are many ways to do this, but the one that currently stands out is JWT (JSON Web Token), mainly because it is safe and easy to implement. JWT is a data transfer system that can be sent via URL, POST, or in an HTTP header. This information is digitally signed, for example, signed with the HMAC algorithm or public/private keys using the RSA algorithm.	X	X	1
	Decentralised authentication		**Centralise authentication**	Single Sign-On	X		0

	Non-secure communication		**Use Transport Layer Security (TLS)**	It enables the Secure Socket Layer (SSL)/Transport Layer Security (TLS) security layer to encrypt the data exchanged between the HTTP agent and the server. Adopt HTTPS, even below the API level. It is very common to see applications with HTTPS only for the external communication layer of the application.	X	X	1
	Publicly accessible microservices	Imagine that our application, which uses the microservices architecture, has three different types of clients. The first is a web frontend, the second is a mobile application, and the third is another service external to our application. Each of these clients expects a different response to their respective requests. Without the gateway, each client should know directly the microservices responsible for delivering the information and know-how to manipulate them.	**Use API Gateway**	If you imagine microservices as a great orchestra, the gateway API would be the concertmaster. The gateway pattern instruments microservices. The API gateway is positioned ahead of microservices. Some benefits we get from adopting the API gateway are optimized endpoints and centralized middleware functionality. Centralized middleware functionality means that levels of security, permissions, authentication, and other validations are at the gateway level.	X	X	1
19	**Non-standard Authorization**		**Use OAuth 2.0**	Setting proper access controls and user authorization should be our first priority. One may use OAuth2 for controlling user authorization.	X	X	1
	Non-standard identity propagation		**Use JSON Web Tokens (JWT)**	Access controls can be set to control the scope of access and permissions for different types of user groups as per your needs. Third party services can also be made use of in this context. JWT or JSON Web Token based authentication may be used, JWT is a good framework in this regard.	X	X	1

	Publicly accessible microservices		**Use API Gateway**	Use of API gateway and isolation of resources. Various 3rd party API gateways can be used to accomplish this purpose. Isolation of APIs and internal components to reduce the exposed attack surface.	X	X	1
	Hardcoded secrets	Never store sensitive keys and other information in Environment variables. These can in certain cases get exposed via application logs, or, can also be accessed unintentionally by other services, which makes it unsafe.	**Encrypt secrets at rest**		X	X	1
	Using experimental solutions	Never use newer experimental encryption algorithms that often come bundled with frameworks, sometimes customized, and they have various kinds of vulnerabilities which may stay unknown or, have lesser-known exploits in them.	**Use already validated solutions**		X	X	1
20	**Decentralized Authentication**	The microservices architecture is a typical distributed system spread over the network, running into multiple services instances and also being accessed thru different clients.	**Use API Gateway**	In order to centralize the access coming from the externals, you should define a single entry point to route all external service clients.	X	X	1
	Non-secure communication		**Use Transport Layer Security (TLS)**	All HTTP connections should be encrypted using TLS (or SSL) protocols. That is going to protect your application context from man-in-the-middle, eavesdropping and tampering attacks providing a bidirectional encryption channel between the client and the server.	X	X	1

Non-encrypting sensitive data	Beyond the encrypted data in transit that you might have defined using HTTPS or any other secure protocol, you also need to ensure the data privacy at the application level. Such as the data you are going to store at the database or even any confidencial data you have defined at the source code level.	**Encrypt data at rest**	All sensitive data should be encrypted as early as possible and decrypted as late as possible.	X	X	1
Own crypto code		**Use already validated solutions**	Using existing data encryption technologies (do not create your own)	X	X	1
Non-proactive security	Nowadays, one of the most common attack at the Internet is the denial of service (DoS), where the hackers try to make the network resources unavailable over flooding the application with requests	**Use attack mitigation techniques**	In order to prevent this issue you should throttle the external requests so then those clients don't consume all the application bandwidth. You can control the rate of traffic sent or received by a particular client checking its IP number and then limit the rate of their requests applying blocking rules by time based period (hourly, daily or monthly) or adding them in a blacklist.	X	X	1
Non-proactive security	Microservices authentication process are pretty much about using access tokens. That means every single request is going to receive a token in order to identify the user access and then give him access to the protected resources.	**Use attack mitigation techniques**	To avoid any vulnerability, such as SQL injections, cross-site scripts you should implement protection using security filters at the API gateway level, such as CSRF and CORS filters.	X	X	1

	unnecessary privileges to microservices	**Least privilege principle**	Using of trusted base container images, limiting permissions to the bare minimum needed, don't running anything using SUDO, applying namespaces and group permissions to isolate access to the resources, never storing secrets at the container disk and limiting available resources consumption are good practices we need to follow in order to assure a minimum level of security at the infrastructure, after all that is going to manage all your runtime environment.	X	X	1
	~~**Not monitoring services**~~	~~**Monitor everything that is possible**~~	~~You should monitor everything that is possible inside your microservices architecture. That is going to detects and prevents possible attacks and ensure you are prepared to support it. And if you are really exposed at some point you can identify and quickly fix it before turns too critical.~~		X	0
21	**unnecessary privileges to microservices**	**Least privilege principle**	Develop an understanding of what access is needed from a minimalist perspective; less is better. Consider what damage can be done to (or by) each particular microservice if it should be compromised, and see what can be adjusted or safeguarded against.	X	X	1
	Non-scalable security controls	**DevSecOps**	Where possible, use centralized security or configuration policies which can be applied across the board to build consistency and reduce the necessity for human analysis or interaction.	X	X	1

	Non-secure communication	**Use Transport Layer Security (TLS)**	When using external access (such as to another server or storage device) utilize encryption for data in transit (such as through HTTPS via certificates) and at rest.	X	X	1
22	**Non-secure communication**	**Use transport Layer Security (TLS)**	Select a security solution that fits the ecosystem already in place without requiring adjustments, such as changing how traffic is routed, the submission of SSL certificates or the alteration of IP addresses. Secure the channels through which the applications are being delivered. That means protecting APIs and web and mobile services from attack vectors such as protocol manipulation, data manipulation in servers, and session and credential attacks.	X	X	1
	Non-scalable security controls	**DevSecOps**	Deliver a security posture that is scalable and elastic to adapt to changing business needs. Automate the monitoring and mitigation of attacks everywhere in the ecosystem to support the continuous deployment process for applications.	X	X	1
23	**No layered security**	**Use defense-in-depth**	Defense-in-depth is a strategy in which several layers of security control are introduced in an application. Sensitive services get layers of security cover, so a potential attacker who has exploited one of the microservices in the application may not be able to do so to another microservice or other layers of the application.	X	X	1

Decentralized Authentication	In a microservices-based application, the proliferation of communication interfaces increases an application's overall attack surface. Each exposed API and communication channel creates a potential attack vector that developers need to make sure they address.	**Use API Gateway**	An API gateway should provide a single point of entry for traffic, which it directs to various microservices. These API gateways often use token-based authentication to manage what data privileges particular services have and dictate how they can interact with that data. Since the clients don't directly access the services, they cannot exploit the services on their own.	X	X	1
Publicly accessible microservices		**Use firewall for the API gateway**	You can add a layer of protection if you place the API gateway behind a firewall. This ensures all the microservices used in a particular application are secure.	X	X	1
Non-secure communication		**Use transport Layer Security (TLS)**	Microservices often need to communicate with each other via data packets and APIs. Maintain security certificates and ensure any data in-transit is encrypted.	X	X	1
unnecessary privileges to microservices		**Least privilege principle**	Also, closely guard API permissions -- only authorized users should have access to the APIs. One way to handle this is using the principle of least privilege to control access to resources. Provide access to a resource only on an as-needed basis.	X	X	1

	Non-scalable security controls	Microservices security best practices aren't just about deploying the right technology. For microservices adoption to be successful, the development and operations teams should converge in line with the concept of DevOps, but they should also have a close relationship with security groups to understand security processes and how to mitigate security risks.	**DevSecOps**	DevSecOps, dictates that developers and operations staff should make security teams part of the initial application design process, rather than only consulted after things go into production. Developers and security staff should work together to determine how to constantly and automatically monitor microservices-based applications for potential threats. Monitoring tools like Prometheus and InfluxDB can help you attain these centralized monitoring capabilities.	X	X	1
24	**Publicly accessible microservices**		**Use API Gateway**	Any organization that is exposing one or more APIs for external access should deploy some form of API gateway. This does not necessarily need to be a specialised device - for many use cases, organizations use an intelligent reverse proxy that allows them to inspect, authenticate, and rate-limit API requests, only admitting requests that meet appropriate criteria and logging all transactions	X	X	1
	Non-secure communication		**Use transport Layer Security (TLS)**	It almost goes without saying that all API traffic must be encrypted using TLS. A good first step is to quickly standardize on using TLS for all internal communications.	X	X	1

	Trust the Network		**Zero Trust Network**	In addition, the internal security of a microservices application should not be forgotten. It's worthwhile to maintain a healthy degree of suspicion of the other components in the application; even if they can be completely trusted now, there's no telling how the client base for the application will grow in the future	X	X	1
	unnecessary privileges to microservices		**Least privilege principle**	Employ a principle of least-privilege for each component within the application - white-list access control, secret protection using revocable tokens, PKI authentication and access control - with particular focus on APIs that can access sensitive data.	X	X	1
	Non-proactive security		**Use attack mitigation techniques**	rate limiting can limit the potential of DoS attacks.	X	X	1
	Non-standard Authorization		**Use OAuth 2.0**	API clients should be authenticated using both an application identifier - and API key or other shared secret - and a user identifier - an SSL certificate or OAuth token. Even anonymous API requests should be required to use a unique user identifier, in order to apply rate limits and to log traffic.	X	X	1
25	**Trust the Network**	Some do work around this by simply trusting the network and avoiding security checks at each microservice. Over time, trust-the-network has become an antipattern, and the industry is moving toward zero-trust networking principles.	**Zero Trust Network**	With zero-trust networking principles, you carry out security much closer to each resource in your network. The zero-trust network approach assumes that the network is always hostile and untrusted, and it never takes anything for granted. Each request must be authenticated and authorized at each node before being accepted for further processing.	X	X	1

Non-standard identity propagation	Nothing is shared among microservices (or only a very limited set of resources), and the user context has to be passed explicitly from one microservice to another. The challenge is to build trust between two microservices so that the receiving microservice accepts the user context passed from the calling microservice. You need a way to verify that the user context passed among microservices isn't deliberately modified.	**Use JSON Web Tokens (JWT)**	Using a JSON Web Token (JWT) is one popular way to share user context among microservices. you can think of a JWT as a JSON message that helps carry a set of user attributes from one microservice to another in a cryptographically safe manner. You have multiple ways to authenticate a system. The most popular options are certificates and JWTs. With JWT, you have an assurance that a man in the middle can't change its content and go undetected because the issuer of the JWT signs it.	X	X	1
Non-standard Authorization		**Use OAuth 2.0**	In these kinds of delegated use cases, in which a system requests access on behalf of another system or a human user, OAuth 2.0 is the de facto standard for security. OAuth 2.0, which is an authorization framework for delegated access control, is the recommended approach for protecting APIs when one system wants to access an API on behalf of another system or a user.	X	X	1

Non-secure communication	When you transfer data from your client application to a microservice or from one microservice to another microservice—depending on the strength of the communication channel you pick—an intruder could intercept the communication and change the data for their advantage.	**Transport Layer Security (TLS) mutual authentication**	The most common way to protect a message for integrity is to sign it. Any data in transit over a communication channel protected with Transport Layer Security (TLS), for example, is protected for integrity. Mutual TLS is another popular way to secure service-to-service communications in a microservices deployment. In fact, this method is the most common form of authentication used today. Each microservice in the deployment has to carry a public/private key pair and uses that key pair to authenticate to the recipient microservices via mTLS. With TLS (one-way), the recipient microservice can't verify the identity of the client microservice. That's where mTLS comes in. mTLS lets each microservice in communication identify the others.	X	X	1

Non-encrypting sensitive data	Along with the data in transit, the data at rest needs to be protected for confidentiality.	**Encrypt data at rest**	To protect a system for confidentiality, both the data in transit and at rest must be protected. The data in transit can be protected with TLS, and data at rest can be protected by encryption. Encryption should also apply to data at rest to protect it from intruders who get direct access to the system. This data can be credentials for other systems stored in the filesystem or business-critical data stored in a database. Most database management systems provide features for automatic encryption, and disk-level encryption features are available at the operating-system level. Also, keep in mind that encryption is a resource-intensive operation that would have a considerable impact on your application's performance unless you find the optimal solution.	X	X	1
No layered security		**Use defense-in-depth**	Defenses against such attacks can be built on different levels. On the application level, the best thing you could do is reject a message (or a request) as soon as you find that it's not legitimate. Having layered security architecture helps you design each layer to take care of different types of attacks and reject an attacker at the outermost layer.	X	X	1

Decentralized security policy management		**Use API Gateway**	The API gateway centrally enforces security for all the requests entering the microservices deployment, including authentication, authorization, throttling, and message content validation for known security threats. API gateway could enforce corporatewide access-control policies, which are probably coarse-grained	X	X	1
Non-secure communication		**Use JSON Web Tokens (JWT)**	JSON Web Token is the third approach for securing service-to-service communications in a microservices deployment. Unlike mTLS, JWT works at the application layer, not at the transport layer. JWT is a container that can carry a set of claims from one place to another. In most cases, JWT-based authentication works over TLS; JWT provides authentication, and TLS provides confidentiality and integrity of data in transit.	X	X	1
Publicly accessible microservices	In a typical microservices deployment, microservices are not exposed directly to client applications.	**Use API Gateway**	In most cases, microservices are behind a set of APIs that is exposed to the outside world via an API gateway. The API gateway is the entry point to the microservices deployment, which screens all incoming messages for security. The key role of the API gateway in a microservices deployment is to expose a selected set of microservices to the outside world as APIs and build quality-of-service (QoS) features. These QoS features are security, throttling, and analytics.	X	X	1

26	**Non-secure communication**	**Use transport Layer Security (TLS)**	One way to limit the opportunities for bad actors is to ensure that all communications in a network segment use SSL/TLS. This provides confidentiality and integrity protection of data in flight, server authentication for clients, and adds important—though optional client-side authentication for servers.	X	X	1
	Centralized authorization	**Use a token-based approach**	Tokens are JSON-based, and protocols are simple to implement as API endpoints. But they also address a deeper concern about the implicit trust a user invests in applications. The new token model maintains that we should never trust a client or a server application with something as powerful as a password (or any primary authentication factor).	X	X	1
	Non-standard Authorization	**Use OAuth 2.0**	OAuth 2.0 is the preferred framework for secure authorization in modern application architectures. What begins as a simple way to delegate authorization between websites is now the primary means of API authorization. OAuth allows users to delegate access between distributed applications. It is not an authentication protocol, which proves a user's claim to an identity. It is an authorization protocol that lets a user (the resource owner) grant an app (the client) access to an API (the resource) on their behalf. This access is for a limited time and with limited scope.	X	X	1

Non-standard authentication	**Use OpenID Connect**	OpenID Connect is an authentication layer built on top of the OAuth framework. OAuth is concerned only with authorization, making no attempts to define how authentication takes place. OpenID Connect takes this on, providing flows to authenticate an end-user and provide claims back to a relying party.	X	X	1
Non-secure communication	**Use JSON Web Tokens (JWT)**	JSON Web Token (JWT) is a simple, JSON-based packaging format for exchanging claims. The claims can be anything you can represent in JSON; JWT adds only a formalized header and body, a signing mechanism (JWS), optional encryption (JWE), and a simple web encoding. The ID Token from OpenID Connect is a JWT.	X	X	1
Trusting your own components	**Zero Trust Network**	a singular approach could be taken to protecting every API endpoint in a microservice architecture, with maximum security using a “zero trust” mentality		X	0
Decentralized security policy management	**Use API Gateway**	Gateways do the same, but operate at a higher level, enforcing sophisticated policies by interpreting application protocols on a transaction-by-transaction basis. They are programmable and usually responsible for authentication, authorization, threat detection, and sophisticated traffic management. API gateways excel at enforcing security policies and accommodating unusual networking challenges.	X	X	1

27	**Decentralized security policy management**	Each microservice can be accessed independently through its own API, and it needs a mechanism to ensure that each request is authenticated and authorized to access the set of functions requested. However, if each microservice performs this authentication individually, the full set of a user's credentials is required each time, increasing the likelihood of exposure of long-term credentials and reducing usability. In addition, each microservice is required to enforce the security policies that are applicable across all functions of the application that the microservice belongs to (such as JSON threat protection for a Node.js application).	**Use API Gateway**	This is where the API Gateway comes in, acting as a central enforcement point for various security policies including end-user authentication and authorization. The API gateway acts as a guard, restricting access to the microservices' APIs. It ensures that a valid access token is present and that all policies are met before granting access downstream, creating a virtual walled garden.	X	X	1
	unnecessary privileges to microservices	A major problem with this design is that the service provider gives access to all data provided by the set of functions that the service account is permitted to use. It does not consider the authenticated user's security context	**Least privilege principle**	This approach is too permissive and is against the principle of least privilege.	X		0
	Non-proactive security		**Use attack mitigation techniques**	Rate-limiting prevents calling of an API more than the allowed number of times for a given period by a specific consumer.	X	X	1
	Non-standard Authorization	It is important to have user-level security context and E2E trust across the entire journey, in addition to service level trust among the microservices of an application.	**Use OAuth 2.0**	You can use protocols such as OpenID Connect, OAuth 2.0, and SAML to facilitate AuthN and AuthZ, and aid in designing a system that handles security at the right place and the right time and guarantees end-to-end trust across the entire journey.	X	X	1

	Non-standard authentication	It is important to have user-level security context and E2E trust across the entire journey, in addition to service level trust among the microservices of an application.	**Use OpenID Connect**	You can use protocols such as OpenID Connect, OAuth 2.0, and SAML to facilitate AuthN and AuthZ, and aid in designing a system that handles security at the right place and the right time and guarantees end-to-end trust across the entire journey.	X	X	1
28	**Non-standard authentication**	Microservices don't lend themselves to the traditional mode of identity control.	**Use OpenID Connect**	By using OAuth with OpenID Connect, and by creating a standards based architecture that universally accepts JWTs, the end result is a distributed identity mechanism that is self contained and easily to replicate. Constructing a library that understands JWT is a very simple task. In this environment, access as well as user data is secured. Creating microservices that communicate well and securely access user information can greatly increase agility of the whole system, as well as increase the quality of the end user experience.	X	X	1

Non-standard Authorization	Microservices don't lend themselves to the traditional mode of identity control.	**Use OAuth 2.0**	By using OAuth with OpenID Connect, and by creating a standards based architecture that universally accepts JWTs, the end result is a distributed identity mechanism that is self contained and easily to replicate. Constructing a library that understands JWT is a very simple task. In this environment, access as well as user data is secured. Creating microservices that communicate well and securely access user information can greatly increase agility of the whole system, as well as increase the quality of the end user experience.	X	X	1
29 Non-proactive security		**Use attack mitigation techniques**	To ensure API security, a WAF solution is needed for inspecting the incoming and outgoing HTTP/HTTPS as with any other web application and provide capabilities such as profiling, blocking attacks, bot and DDoS protection, preventing account takeover and more. Your WAF should also help secure applications and data in the new application environment, with automatic deployment anytime new services or containers are provisioned.	X	X	1

Non-scalable security controls		**DevSecOps**	The operational aspects of managing your WAF solution should be automated and templatized, such that your security can be easily scaled out. So, regardless of whether you spin up a new server, deploy a new application, or move an existing service from one server to another, the security policies and provisioning layer linked to that service are automatically deployed.	X	X	1
Non-secure communication		**Use transport Layer Security (TLS)**	Your organization should continue to focus on: Ensuring confidentiality by encrypting communications and data at rest.	X	X	1
Non-encrypting sensitive data	Data security becomes even more critical as the applications and infrastructure become more distributed, with complex interdependencies that potentially span services, APIs, containers and clouds.	**Use a data-centric audit and protection (DCAP)**	A DCAP solution helps you protect data in databases, file stores, and big data repositories with real-time monitoring, auditing, and security and rights management. With a DCAP solution, you can: - Analyze all database activity in real time. You can monitor all users who access the database, whether through a browser, a mobile app or a desktop application. - Take action to avoid compromise and data loss, such as blocking access to sensitive data based on security policies.	X	X	1

30	**No layered security**	With the third-party APIs used by your application, you may not even be aware of all the risks because you may not know all the parameters and supported endpoints (documentation is never perfect!) or because something has changed during the last version update.	**Use defense-in-depth**	API's should be monitored for security issues at their respective ingress points. The right API security solution should be able to understand east-west microservices' protocols' syntaxes and, thus, to detect injections such as the Open Web Application Security Project (OWASP) Top-10 A1 class injection risks.	X	X	1
	Unauthenticated API requests		**Use Authentication**	During the design time, the most important issue is the correct implementation of API authentication and credential management. Lack of robust authentication practices can have scary consequences.		X	0
31	**Non-scalable security controls**		**Continuous Security Testing**	Automating security testing in CI/CD leverages existing testing developers are already doing. Automated testing can create and update baselines before every release with OWASP Top 10 and advanced libraries that are informed by discovered abnormalities across clients. Through automation, developers can meet their responsibility to facilitate fast releases and quicker code fixes. It releases teams from the impracticality of safer, more homogenous coding or heavy training.	X	X	1

	Unauthenticated API requests		**Use Authentication**	Access is another critical factor to microservices security. Secure access to microservices with the API access control. This is absolutely fundamental to truly securing applications that consist of microservices software. There are multiple, independent API services that require additional tools to manage API access control. Be thorough with each tool you use	X	X	1
	No layered security	Protecting your microservices will never come down to a singular solution or practice. You need to think like a security expert. Think: defense in depth.	**Use defense-in-depth**	The defense in depth approach creates a multi-layer security to prevent attacks. Defense in depth includes security measures such as: - filtering communication flows; - authentication authorization; - access controls for microservices; and - using the encryption technologies specific to a group.	X	X	1
	Non-scalable security controls	You need to Integrate automated security testing into your build or continuous integration process (CI/CD).	**DevSecOps**	Automation is the key to integrating quality protection in a way that ensures quick feedback on the impact of new changes. Continuous security testing for continuous software development helps realize the speed and the flexibility and ensures the faster recovery.	X	X	1
32	**Decentralized authentication**		**Use API Gateway**	Services require authentication and authorization. In a microservices world you typically have a front-end gateway that manages connections from the outside world which then connect you to back-end microservices to handle the request.	X	X	1

Centralized authorization	Approach 1: Global Authentication and Authorization If your back-end (micro) services are to have total faith that the front-end (global) services have authenticated and authorized the transaction, then you can go for option 1. The problem with this is that when you push authorization into the global context, you are moving conceptual business logic from your back-end microservices to the front-end — and that is often the wrong move to make.	**Decentralize authorization**	This approach is likely the best fit for most people. Keep the clunky authentication layer at your front-end global services layer. Then, when the front-end calls the back-end microservices to do an actual job, it can provide a security context. This allows the microservices to not care how someone is authenticated, but is still able to maintain the business logic decision making of what actions this security context is allowed to perform.	X		0
Decentralized authentication	Approach 4: Service Authentication and Authorization You can do it, but it is really a bad violation of good layered architectures to not do the authentication globally. If you have a small set of services, this is probably not big problem — but once you grow past a certain size you really want your services to deal with only what they need to deal with.	**Use API Gateway**		X		0
Non-secure communication	If any service within your microservices mesh is able to call you and provide a faked security context… That is just a recipe for disaster.	**Transport Layer Security (TLS) mutual authentication**	You need to have strong authentication and authorization between your services (by using something like Istio and its mTLS). You must be certain that when you get a request to do something in your microservice with a given security context that the security context can be trusted.	X	X	1

33	**Non-proactive security**	Ensuring that your applications are secure is no easy feat, and managing a number of services that have multiple entry points from the outside can be difficult. As the number of services grow, the magnitude of this issue is amplified.	**Use attack mitigation techniques**	Setting up a firewall application or an alternative solution in front of your system can correct a problem like this by ensuring that only the appropriate traffic arrives at your application's front door and that it does not contain malicious codes or threats.	X	X	1
	Non-secure communication		**Use transport Layer Security (TLS)**	Traffic between your microservices should be encrypted on the cloud. This means that, in addition to your microservices handling encrypted traffic, they will also need to ensure that the performance of your underlying applications does not suffer as a result of the extra work they have to perform with encrypting and decrypting information.	X	X	1
	Hardcoded secrets	Embedding secrets into your applications is a very bad idea. Best practices for modern architecture strongly advise against storing any credentials on your servers. Of course, this brings up the question of how you will allow applications to authenticate with each other and third-party services if the credentials cannot be stored locally?	**Use third-party authentication**	One strategy is to use third-party tools or the tools and services that are already available from most cloud providers. The concept is pretty simple. When you initiate an authentication request, you ask another service to request a temporary set of credentials on your behalf, which allows you access for a set period of time. This solves the issue of longevity of credentials because they expire after a certain period of time, and because there are no credentials that are embedded in the microservice itself.	X	X	1

	Non-scalable security controls	When you are dealing with microservices, there can be several changes each day. When you upgrade your application by changing or adding functionality, you will need to ensure that your code is (at minimum) the same as it was before, if not even better.	**Continuous Security Testing**	This requires scanning the added code for vulnerabilities and weaknesses before the code is even deployed. You will need to tie this into your continuous integration processes so that this is performed as part of your release process.	X	X	1
34	**Non-proactive security**		**Use attack mitigation techniques**	We also recommend enabling a feedback loop to provide alerts from the middle tier and backend service to your WAF. This will help the WAF know when to block these attacks. A WAF should also monitor the volume of cache misses. If an API gateway is constantly performing middle tier service calls due to cache misses, that suggests that the cache is not configured correctly or potential malicious behavior.	X	X	1
	Decentralized authentication		**Use API Gateway**	It may be possible for your middle tier and backend services to limit the batch or object size requested. This can also be done in the client code, and potentially even enforced in the API gateway. API gateways and other microservices should prioritize authenticated traffic over unauthenticated traffic.	X	X	1
	Non-proactive security		**Use attack mitigation techniques**	Ensure you have reasonable client library timeouts and circuit breakers.	X	X	1

	Not knowing your system	Application layer attacks focus on expensive API calls, using their complex interconnected relationships to cause the system to attack itself — sometimes with a massive effect. In a modern microservice architecture this can be particularly harmful.	**Identify critical services**	First and foremost it is critical to know your system. You should understand which microservices impact each aspect of the customer experience. Look for ways to reduce inter-dependencies on those services. If one service becomes unstable the rest of your microservices should continue to operate, perhaps in a degraded state.	X	X	1
35	**Trust the Network**	A network, private or public, is never secure, period. Network perimeter security is no longer the advised best practice.	**Zero Trust Network**	Zero Trust = Authentication + Authorization + Encryption. In Zero trust the attention is turned towards directly protecting partitioned resources against explicitly untrusted clients. In other words, a Zero Trust system does not differentiate on where you are, but only cares about who you are. Having a reliable way to prescribe role based access controls severely limits the damage a malicious or negligent client can do.	X	X	1
	Non-secure communication	Assuming resources are protected, network traffic still is not. There cannot be an assumption of trusted communication between any parties, regardless of where it originated.	**Use transport Layer Security (TLS)**	Be it user to service, service to user, service to service, or user to user, all communication, machine or human, must be encrypted and protected from end to end.	X	X	1
	Not knowing your system		**Identify critical services**	All critical data stores, applications, assets, and services must be catalogued. This can range from SaaS providers, to web-connected devices, customer information stores, identity providers, or any other resource that contributes to infrastructure or operations.		X	0

36	**Hardcoded secrets**	Don't put secrets in native apps!	**Encrypt secrets at rest**	An example of this is Proof-Key for code exchange (PKCE), PKCE replace the need for clients secrets. Hashed secrets.	X	X	1
	Trusting your own components		**Zero Trust Network**	Treat components of your application the same way you'd treat third-party applications.	X	X	1
	Non-standard Authorization	How can I let an app access my data without giving it my password?	**Use OAuth 2.0**	OAuth acts as a buffer between giving your password to an App and then giving it just the data that it wants.	X	X	1
37	**Decentralized authentication**		**Use API Gateway**	Even with IP whitelisting in place, having an API gateway in place is still best practice. This aids in authentication and ensures the backend is only receiving properly formed API calls.	X	X	1
	Non-standard Authorization		**Use OAuth 2.0**	The most common is OAuth and OAuth2 for communicating and securing communications between APIs. Underneath is token-based and claims-based authentication where the APIs are passing digitally signed tokens back and forth to verify the token is representative of who is making the call.	X	X	1
	Unauthorized API requests	authentication alone is not enough to grant access to an API, there should be an authorization step to determine what resources have access to the API.	**Use authorization**	There various ways to check for proper authorization include content-based access control (CBAC), role-based access control (RBAC) or policy-based access control (PBAC) — these methods ensure business data remains fully protected against unapproved access.	X	X	1

Non-proactive security		**Use attack mitigation techniques**	Use rate-limiting API calls to mitigate distributed denial-of-service (DDoS) attacks and protecting the backend applications that process the API calls. Securing the APIs themselves by applying a rate-limiting policy that sets a threshold on the number of requests the API gateway accepts each second (or other time period) from a specified source, such as a client IP address.	X	X	1
Non-encrypting sensitive data		**Encrypt data at rest**	Encryption in transit is great, but not enough. It should be enforced all the way, including while the data is at rest. Enforce Encryption all the way	X	X	1
~~**Not monitoring services**~~		~~**Monitor everything that is possible**~~	~~alert and monitoring are key to see what is going on in your world to protect against bad behaviors. Logging and auditing should be used proactively in detecting and alerting you against threats and attack.~~	X		0
Non-secure communication	Protecting the backend applications using HTTPS — HTTPS protocol should be used traffic between API gateway and the backend systems that process the API requests.	**Transport Layer Security (TLS) mutual authentication**	When it comes to securing APIs, serving them via HTTPS is a basic yet absolute requirement. It's relatively straightforward to setup: the tools and processes involved are widely available and well understood. Make sure traffic is secure and encrypted. Make sure the client and server are both SSL.	X	X	1

38	**Decentralized security policy management**		**Use API Gateway**	API Gateways are a natural integration points because they sit at the place where there is a clear contract between the consuming entity, the client and the API which means in turn we can make sense of that information and derive meaningful authorizations.	X	X	1
	Non-standard Authorization	When we talk about microservices, we talk about dynamic authorization	**Use OAuth 2.0**	Use OAuth 2.0 to manage Authorization. OAuth is use manage delegated authorization. OAuth is use to solve the password antipattern.	X	X	1
	Non-standard authentication		**Use OpenID Connect**	Use OIDC to manage Authentication. OIDC added an identity layer to OAuth 2.0 to more easily facilitates single sign-on across domains. OIDC introduced different kinds of token formats like JWT.	X	X	1
39	**Centralized authorization**	In MSA, we can replace the identity server with a microservice, which, in my opinion, leads to a big complicated dependency graph.	**Use a token-based approach**	The client talks to an identity or SSO server, authenticates itself, receives a signed token that describes the user and his roles with SAML or OpenIDConnect, and sends the token to microservices with each request. Each microservice verifies the token and authorizes the calls based on the user roles described in the token. This model pushes the authentication to the client and does access control at microservices while simplifying dependencies.	X	X	1

40	**Non-secure communication**		**Use transport Layer Security (TLS)**	Developers should strive to use isolation and API security measures—such as transport layer security (TLS) encryption and strong API keys—to create secure communication and authentication among services.	X	X	1
	Non-proactive security	The technology stack and the resulting dependencies are vastly multiplied in a microservice environment compared to a monolithic environment."	**Keep third-party components up to date**	The most popular platforms for cloud-based services are based on open-source components and third-party libraries. Developers who use these stacks need to assiduously update and maintain each microservice. Overall, it's a lot more fragile when you consider the possibility of exploits or flaws creeping in as a part of the stack or the third-party components that you are using.	X	X	1
	Own crypto code	Developers should not try to create the fundamental security building blocks.	**Use already validated solutions**		X	X	1
	Non-scalable security controls	Developers cannot afford to have any manual processes for checking and validating security, because changes are deployed so frequently.	**Continuous Security Testing**	Security also has to move as quickly as the agile development process, which means that it is absolutely critical that security processes are integrated into the CI/CD pipeline. You should add automated scanners to your process to check, as often as possible, if there are any regressions or unexpected behaviors. You need a global view to understand the architecture of the applications, and what vulnerabilities need to be fixed.	X	X	1

42	**Decentralized Authentication**		**Use API Gateway**	API gateways are the most commonly used solution, and with good reason – they provide many great out-of-the box management services in addition to security. One single point of entry with API gateways and security platforms. You need a robust, centralized authentication and authorization policies for access.	X	X	1
43	**Non-standard Authorization**	Microservices should only be invoked after requesting authentication and, ideally, authorization if levels of privileges are available.	**Use OAuth 2.0**	OAuth (currently in version 2.0) and OpenID Connect are frameworks that lend themselves to typical implementation of microservices that use RESTful APIs. In essence, an access token is issued by an authorization server to a trusted client application. OpenID Connect is built on the top of OAuth 2.0 and uses JSON Web Tokens as an identity token.	X	X	1
	Centralized authorization	Verifying the access token at the gateway level makes it vulnerable to the confused deputy problem. This comes from microservices trusting the gateway based on its mere identity (sometimes even an IP address), which makes it open to misuse if compromised.	**Use a token-based approach**	Having access control enabled and scopes of the access token checked by microservices prior to responding to a request is a possible mitigation. Note that having a dedicated service acting as an authorization server provides three main benefits: decoupling and isolation in case the system is compromised, separation of concerns, and an auditing point.	X	X	1

	Decentralized Authentication	Microservices require a central service (the conductor) to send requests and organize the workflow.	**Use API Gateway**	A common model for microservices uses API gateways. Being a dedicated element that does not directly participate in the application itself, it can also act as an intrusion detection system (IDS). Orchestration is normally executed at the gateway level, which is a single entry point to the system, making it ideal for storing logs and auditing tasks.	X	X	1
	Non-scalable security controls		**Continuous Security Testing**	Automated testing and verification become crucial as typically these applications are developed using agile methodologies and rely on fast iteration cycles. The deployment of the application should be along with security tests and verification tools, which should be continuous and periodic and include vulnerability management.	X	X	1
	Non-proactive security		**Use attack mitigation techniques**	Any request from the outside world must pass through a firewall and IDS, and container firewalls should inspect requests from the gateway or any potential internal traffic.	X	X	1
44	**Non-secure communication**	Classic attacks on the network stack and protocols; attacks against protocols specific to the service integration style (SOAP, RESTful Web Services). Attacks include eavesdropping (sniffing), identity spoofing, session hijacking, Denial of Service (DoS), and Man-in-the-Middle (MITM); also attacks on TLS: Heartbleed and POODLE.	**Use transport Layer Security (TLS)**	Use of standard and verified security protocols such as TLS or JSON security standards.	X	X	1

Non-secure communication		**Transport Layer Security (TLS) mutual authentication**	Although there are currently few industry practices for microservice security, some interesting trends present themselves. The first one is the use of Mutual Transport Layer Security (MTLS) with a self-hosted Public Key Infrastructure (PKI) as a method to protect all internal service-to-service communication. TLS with mutual authentication addresses problems of service authentication and traffic encryption.	X	X	1
No layered security	Until recently perimeter defense was the most common approach to security of microservice-based systems. From the modern security perspective, perimeter security is in general considered insufficient.	**Use defense-in-depth**	We should rather assume that the other services in the system may be compromised and hostile (“trust no one”). The rise of microservices, as well as advances in security automation, facilitate placement of additional security mechanisms inside the perimeter. In other words, defense in depth as a concept of placing multiple and sometimes overlapping security mechanisms on different levels throughout the system becomes more feasible with microservices.	X	X	1

Trusting your own components	We assume an adversary can compromise at least one service inside the perimeter and wants to move laterally through the system to gain full control. If internal services blindly trust whoever is calling them, then a single compromised microservice will allow an attacker to manipulate all the other nodes in the microservice network, for example, by issuing arbitrary malicious requests that the nodes will fulfill. The latter is sometimes referred to as a confused deputy problem. The adversary can attempt to eavesdrop on the inter-service communication, insert and modify data in transit.	**Zero Trust Network**		X	X	1
Centralized authorization	Microservices should be aware of the user authentication state, i.e. whether the user was authenticated, and what the user's role is in authorization context. The user needs to be identified multiple times in each service down the operation chain, as each service calls other services on the user's behalf.	**Use a token-based approach**	Token-based authentication is a well known commonplace security mechanism that relies on cryptographic objects called security tokens containing authentication or authorization information. A security token is created on the server-side upon the successful validation of the client's credentials and given to the client for subsequent use. Security tokens substitute the client's credentials within a limited time-frame.	X	X	1

45	**Trusting your own components**	Trustworthiness is also an issue when dealing with microservices. For example, an adversary could compromise or gain control of a component, which isn't uncommon within the public cloud context. However, in a typical microservice architecture, the other components assume a trusted component base; thus, there's a real risk that attacks can be easily propagated due to the microservices' dependencies and (blind) trust of peer components.	**Zero Trust Network**		X	X	1
	Non-secure communication	In the absence of a centralized orchestrator, for example, microservices coordinate among themselves through message-based communications. These communications, if not protected, won't let us achieve secure guarantees during data exchange and/or receive falsified or modified coordination which is required when undertaking complex and elaborate functionalities.	**Use transport Layer Security (TLS)**		X	X	1
46	**Non-scalable security controls**	Any vulnerability should be identified early in the development lifecycle and should have shorter feedback cycles.	**Continuous Security Testing**	A proper secure development lifecycle and test automation strategy needs to be there to make sure that we do not introduce security vulnerabilities at the code level. We need to have a proper plan for static code analysis and dynamic testing — and most importantly those tests should be part of the continuous delivery (CD) process.	X	X	1

Non-secure communication	Securing service-to-service communication is the most critical part of securing microservices.	**Use JSON Web Tokens (JWT)**	JWT (JSON Web Token) defines a container to transport data between interested parties. A JWT can be used to transfer data securely between interested parties over an unsecured channel. A JWT can be used to transfer signed and/or encrypted messages. A JWT can be signed or encrypted or both. A signed JWT is known as a JWS (JSON Web Signature) and an encrypted JWT is known as a JWE (JSON Web Encryption).	X	X	1
Non-secure communication	Securing service-to-service communication is the most critical part of securing microservices.	**Transport Layer Security (TLS) mutual authentication**	Mutual authentication authenticates both parties—the client and the server. In a microservices environment, TLS mutual authentication can be used between microservices to authenticate each other. With TLS mutual authentication, the end-user identity has to be passed at the application level—probably as an HTTP header.	X	X	1
Decentralized Authentication		**Use API Gateway**	With the API gateway pattern the microservices, which need to be exposed outside, would have a corresponding API in the API gateway. Not all the microservices need to be exposed from the API gateway. The end user's access to the microservices (via an API) should be validated at the edge — or at the API gateway.	X	X	1

	Non-standard Authorization		**Use OAuth 2.0**	The most common way of securing APIs is OAuth 2.0. Over time, OAuth 2.0 has become the de-facto standard for API security. Whoever wants to access a microservice via the API gateway must get a valid OAuth token first.	X	X	1
	Centralized authorization	An authorization check evaluates whether a given user has the minimum set of required permissions to access a given resource. The resource can define who can perform and which actions they can perform.	**Use token-based approach**	XACML is the de-facto standard for fine-grained access control. It introduces a way to represent the required set of permissions to access a resource, in a very fine-grained manner in an XML-based domain-specific language (DSL).		X	0
47	**Non-secure communication**	HTTP transfers data in plain text, but data transfer over the internet in plain text is not a good idea at all. It makes hacker's jobs easy and allows them to get your private information, such as your user ID, passwords, and credit card details easily using a packet sniffer.	**Use transport Layer Security (TLS)**	We definitely don't want to compromise user data, so we will provide the most secure way to access our web application. Therefore, we need to encrypt the information that is exchanged between the end-user and our application. We'll use Secure Socket Layer (SSL) or Transport Security Layer (TSL) to encrypt the data.	X	X	1
	Non-standard Authorization	Providing authentication and authorization is de facto for web applications.	**Use OAuth 2.0**	OAuth is an open authorization mechanism, implemented in every major web application. Web applications can access each other's data by implementing the OAuth standard.	X	X	1

48	**Non-proactive security**		**Use attack mitigation techniques**	A web application firewall (WAF) is configured as your second or third line of defense in a system. To understand what a WAF is, let's look at the definition from the Open Web Application Security Project (OWASP): "A web application firewall (WAF) is an application firewall for HTTP applications. It applies a set of rules to an HTTP conversation. These rules cover common attacks such as cross-site scripting (XSS) and SQL injection.	X	X	1
	Non-secure communication		**Use transport Layer Security (TLS)**	TLS or Transport Layer Security no longer adds any overhead due to the advances in processing power available to servers these days. In addition to this, services inside a firewall generally have a limited number of connections; so, to improve the time that is lost by the TLS handshake, you can use persistent reusable connections in your service to minimize this problem.	X	X	1
	Non-encrypting sensitive data	Assuming our system had been attached to a database for storing things such as user accounts, the attacker would have been able to get access to the complete database of passwords. One of the things that we should think about when are storing data in a database is the encryption of our data.	**Encrypt data at rest**	One of the many benefits microservices give us is that we separate functions and data between our systems. This can make deciding what data to encrypt easier as rather than attempting to understand which data to encrypt within a datastore, you make a simpler decision: is there any data that needs to be encrypted inside this datastore? If so, then simply encrypt all of it.	X	X	1

unnecessary privileges to microservices	**Least privilege principle**	There is a security principle called the least privilege; this recommends that accounts and services have the least amount of privilege to perform their business function. Even if you have ensured that the machine-to-machine communication is secured and there are appropriate safeguards with your firewall, there is always an opportunity for an attacker to access your systems by the back door.	X	X	1
Centralized authorization	**Use JSON Web Tokens (JWT)**	A JSON Web Token (JWT) is a standard for safely passing claims or data attributed to a user within an environment. It is an incredibly popular standard and is available for just about every major language and framework. There are two main strengths of JWT. One is a standard format for the claims, which makes the availability of reliable frameworks possible. The other is the use of asymmetric encryption, which means that because a token is signed, the receiver only needs the public key of the signer to validate that the token has indeed come from a trusted source and this allows us to lock down access to the private keys to an authorization server.	X	X	1

Non-proactive security	**Keep third-party components up to date**	One important element of keeping your system secure is making sure you keep it up to date with all the latest security patches. This approach needs to be applied to your application code and your server's operating system and applications, and if you are using Docker, you also need to ensure that your containers are up to date to ensure you are free from vulnerabilities.	X	X	1
Decentralized Authentication	**Use API Gateway**	In addition to a WAF, an API Gateway can be a useful tool to have; this can serve a dual purpose of routing your public APIs to their backend services and some additional features such as token validation at the edge and input validation and transformation. API Gateways often implement many other first-line features such as but not limited to the following: - Request validation - Authorization - Rate limiting - Logging - Caching - Request and response transformations	X	X	1

49	**Non-standard Authorization**	In a microservice-based architecture, the individual microservices should not perform authentication. It does not make much sense for each microservice to validate the user name and password. For authentication a central server has to be used. For authorization an interplay is necessary: often there are user groups or roles that have to be centrally administered.	**Use OAuth 2.0**	One possible solution for this is OAuth2. There are numerous libraries for practically all established programming languages that implement OAuth2 or an OAuth2 server. Between the microservices, only the access token still has to be transferred. This can occur in a standardized manner via an HTTP header when REST is used. In the case of different communication protocols, similar mechanisms can be exploited. Also in this area, OAuth2 hardly limits the technology choice.	X	X	1
	Non-secure communication		**Transport Layer Security (TLS) mutual authentication**	The communication between the microservices can be protected by SSL/TLS against wiretapping. All communication is then encrypted. Infrastructures like REST or messaging systems mostly support such protocols.	X	X	1
	No layered security		**Use defense-in-depth**	Firewalls can be used to protect communication between microservices. Normally firewalls secure a system against unauthorized access from outside. A firewall for the communication between the microservices prevents that all microservices are endangered if an individual microservice has been successfully taken over. In this way, the intrusion can be restricted to one microservice.	X	X	1

50	**Trust the Network**	Our first option could be to just assume that any calls to a service made from inside our perimeter are implicitly trusted. Depending on the sensitivity of the data, this might be fine. Some organizations attempt to ensure security at the perimeter of their networks, and therefore assume they don't need to do anything else when two services are talking together. However, should an attacker penetrate your network, you will have little protection against a typical man-in-the-middle attack.	**Zero Trust Network**		X	X	1
	Centralized authorization	There is a type of vulnerability called the confused deputy problem, which in the context of service-to-service communication refers to a situation where a malicious party can trick a deputy service into making calls to a downstream service on his behalf that he shouldn't be able to.	**Use a token-based approach**		X	X	1
	Non-encrypting sensitive data	Many of the high-profile security breaches involve data at rest being acquired by an attacker, and that data being readable by the attacker. This is either because the data was stored in an unencrypted form, or because the mechanism used to protect the data had a fundamental flaw.	**Encrypt data at rest**	For encryption at rest, unless you have a very good reason for picking something else, pick a well-known implementation of AES-128 or AES-256 for your platform. Encrypt data when you first see it. Only decrypt on demand, and ensure that data is never stored anywhere.	X	X	1

	Own crypto code	The easiest way you can mess up data encryption is to try to implement your own encryption algorithms, or even try to implement someone else's. Badly implemented encryption could be worse than having none, as the false sense of security (pardon the pun) can lead you to take your eye off the ball. If there is nothing else you take away from this chapter, let it be this: don't write your own crypto. Don't invent your own security protocols.	**Use already validated solutions**	Whatever programming language you use, you'll have access to reviewed, regularly patched implementations of well-regarded encryption algorithms. Use those! And subscribe to the mailing lists/advisory lists for the technology you choose to make sure you are aware of vulnerabilities as they are found so you can keep them patched and up to date. They are good enough! Reinventing the wheel in many cases is often just a waste of time, but when it comes to security it can be outright dangerous.	X	X	1
	No layered security		**Use defense-in-depth**	As I've mentioned earlier, I dislike putting all your eggs in one basket. It's all about defense in depth.	X	X	1
51	**Publicly accessible microservices**	"Before you had to secure one door, now you have 20."	**Use firewall for the API gateway**	To tackle this challenge, it would be beneficial to follow existing reference implementations or use the API gateway pattern.	X	X	1
52	**Non-standard Authorization**	In a monolithic web application, authentication happens with a login form, and once the user is identified, a cookie is set and used for all subsequent requests. In a microservice-based architecture, we can't use that scheme everywhere because services are not users and won't use web forms to authenticate. We need a way to allow or reject a call between each service automatically.	**Use OAuth 2.0**	The OAuth2 authorization protocol gives us the flexibility to add authentication and authorization in our microservices, that can be used to authenticate both users and services. The core idea of OAuth2 is that a centralized service is in charge of authenticating a caller, and can grant some access in the form of codes or tokens; let's call them keys. Those keys can be used by users or services to access a resource, as long as the service providing that resource accepts that key.	X	X	1

Centralized authorization	When a service wants to get access to another service it has to be without any user intervention.	**Use a token-based approach**	A token is usually built as a self-contained proof that you can use a service. Self-contained means that the service will be able to validate the token without having to call an external resource, which is an excellent way to avoid adding dependencies between services. Depending on the implementation, a token can also be used to access different microservices. OAuth2 uses the JWT standard for its tokens.	X	X	1
Centralized authorization		**Use JSON Web Tokens (JWT)**	OAuth2 uses the JWT standard for its tokens.	X		0
Non-proactive security	When you're exposing HTTP endpoints to others, you are expecting callers to behave as intended. Each HTTP conversation is supposed to follow a scenario that you have programmed in the service. In the real world, that's not always the case. If the caller has a bug or is just not calling your service correctly, the expected behavior should be to send back a 4xx response and explain to the client why the request was rejected. That's also the case for malicious requests sent by attackers. Any unintended behavior should be dismissed.	**Use attack mitigation techniques**	Web Application Firewalls (WAF) that can be used to avoid a lot of attacks. Adding protection on the server-side to back-off such zealous clients is usually not hard to do and goes a long way to protect your microservice stack.	X	X	1
Non-proactive security		**Use attack mitigation techniques**	Rate limiting consists of counting how many requests a server is accepting in a period of time and rejecting new ones when a limit is reached.	X	X	1

53	**No layered security**	- Are you just protecting your system at the Internet boundary? - What protections do you have in place if an intruder gets inside your core network? - If you assume that your gateway services have been fully breached, what would you do differently elsewhere? - If your gateway services were fully breached, what data could be gleaned from memory? - If your gateway services were fully breached, what data could be captured from the network traffic? - Are you constantly asking yourselves: “What if this control fails? What’s the next control?	**Use defense-in-depth**	X	X	1
	Non-secure communication	- How easily could someone inside your network get access to the traffic between your services? - Have you upgraded your TLS implementations to the latest versions possible? - Have you configured TLS to eliminate downgrade and weak cipher attacks? - Who on your staff knows everything about TLS and how to configure it safely?	**Use transport Layer Security (TLS)**	X	X	1

Trusting your own components	- Do your services trust each other too much? Or… Do your services trust whoever is calling them too much? (Are you sure only your services can call into your services?) - Does the rest of your system trust your gateway services too much? - Does your web app design treat the browser as an insecure environment? - Does your native mobile app design treat the device as an insecure environment?	**Zero Trust Network**	X	X	1
Non-standard Authorization	-Can your services request any data from each other, or only the data of a user that has given their authority? - If an attacker owned a service, could they pretty easily request anything from its downstream services? - Do your services let their callers access all the APIs that a service offers, or just the ones it needs to fulfil its function?	**Use OAuth 2.0**	X	X	1
Non-standard authentication	- I have a list of cracked passwords and user emails. Could I use your password reminder URL to test which users are in your system? - Do you lock an account after some number of failed login attempts?	**Use Authentication**	X	X	1

unnecessary privileges to microservices	- Do you share a single database login across all your services? - How much data do your services have access to? All of it? Or only what they need? - If an attacker got hold of one service's database credentials, how much data would they get access to? - Do your DB authorisation policies allow updates and deletes to tables that the application only ever inserts into? - Do you share a single messaging middleware login across all your services? - Does your messaging middleware even have login credentials? - Do your services have access to all messages in your system, or only the ones they need to see? - Can your services send messages to any queue, or only where the ones they need to? - If an attacker got hold of one service's messaging credentials, how much data could they get access to? - If an attacker got hold of one service's messaging credentials, what operations could they initiate?	**Least privilege principle**		X	X	1
54 Centralized authorization	The most important aspect of securing communication between application components is preventing unauthorized access, which usually involves the processes of authentication and authorization.	**Use a token-based approach**	An access key, or access token, is transmitted with each request, and only if a known and correct key is passed, access is granted to the service. Token-based authentication and authorization was used to connect to the database servers.	X	X	1

55	**Non-secure communication**	The communication channel between microservices should be secure. Communication between microservices and service registry should use a secure channel.	**Use transport Layer Security (TLS)**	A secure channel should be used for communications. Transport-layer security offers secure point-to-point communication channels. Suggested protection measures: Use transport-layer security.	X	X	1
	Unauthenticated API requests	A weak set of APIs exposes microservices to a variety of security attacks that may result in tampering with data, information disclosure, denial of service and elevation of privileges.	**Use Authentication**	Use keys or security tokens or passwords to protect API. Only authenticated users should access the API. The API should validate all requests.	X	X	1
	Decentralised authentication		**Single Sign-on**			X	0
	Centralized authorization		**Use a token-based approach**	REST web services can use JSON Web Tokens (JWT) as the format for security tokens for authentication and ensuring message integrity. The microservices composition should use multi-factor authentication at all entry points. Any credentials used in the microservices composition should be rotated periodically.	X	X	1
56	**Decentralized security policy management**	A further requirement is to decouple the control of the microservice from the service itself.	**Use API Gateway**	We approach this by designing our architecture using reusable and configurable gateways at the level of each microservice. These components can be added to secure primitive services, and modified to meet different policies. In order for a request to reach a microservice, security policies enforced by the gateway have to be met by the requesting service or party (the consumer microservice); note that the consumer microservice should have another gateway to enforce access control policies.	X	X	1

	Non-standard Authorization		**Use OAuth 2.0**	Open Authorization 2 (OAuth 2) is one of the most commonly used mechanisms in a microservice architecture for access delegation. OAuth 2 access scopes are used to define the token holder's access rights.	X	X	1
	Non-standard authentication		**Use OpenID Connect**	OpenID Connect, built on top of OAuth 2, is commonly used for authentication with MSA; it is an enabler for identity federation by producing an ID token with end-user information, and practice of the separation of concerns principle.	X	X	1
	Centralized authorization	On the other hand, we have the confused deputy problem. As explained, this consists of a component that has access to sensitive resources, and which can be manipulated by an adversary to have indirect access to these resources.	**Use a token-based approach**	The key point to prevent this is to have the resource services, the Department of Justice and Interior Affairs microservices in our scenario, verify that the calling microservice is acting truthfully on behalf of the user. This requires, for example, tokens to be individual to each component, and have finer granularity reflecting users' consents on access rules.	X	X	1
58	**Non-standard Authorization**	Many security analysts do not prefer starting from scratch and recommend using OAuth2 and OpenID Connect to delegate authorization management to a third party or a single (internal) authentication service	**Use OAuth 2.0**	Using libraries and functions can shorten the development time and make it easier. By the same token, several solutions for improving the security level of your OAuth-based authorization service have already been built by some of the biggest companies and smartest engineers around.	X	X	1

	No layered security	You need to identify your most sensitive services, and manually apply a number of different layers of security to them, so that it gets harder for a potential attacker who is able to exploit one of your security layers.	**Use defense-in-depth**	Microservices makes it easier to adopt this strategy in a very microscopic and strategic way—by focusing your security efforts and resources on specific microservices. The architecture diversifies the layers of security you wish to adopt on each microservice. By this, an attacker who is able to exploit one of your services may not necessarily be able to figure out how to exploit the second one.	X	X	1
	Own crypto code	It is advised that when it comes to security you shouldn't try to roll your own new solutions and algorithms unless you've got strong and specific reasons to	**Use already validated solutions**	you've got people skilled enough to create something nearly as good as the open source tools already available	X	X	1
	Publicly accessible microservices	Get your containers out of the public network. An API gateway establishes a single entry point for all requests coming from all clients. It subsequently knows how to provide an interface for all of your microservices.	**Use API Gateway**	By using this technique you can secure all of your microservices behind a firewall, allowing the API gateway to handle external requests and then talk to the microservices behind the firewall.	X	X	1
	Non-scalable security controls		**Continuous Security Testing**	The best solution for Microservices Security is continuous security that is as flexible and agile as your development	X	X	1
59	**Non-proactive security**	Third-party dependencies make up 80% of the code you deploy to production. Many of the libraries we use to develop software depend on other libraries. Transitive dependencies lead to a (sometimes) large chain of dependencies, some of which might have security vulnerabilities.	**Keep third-party components up to date**	You can use a scanning program on your source code repository to identify vulnerable dependencies. You should scan for vulnerabilities in your deployment pipeline, in your primary line of code, in released versions of code, and in new code contributions.	X	X	1

Non-secure communication		**Use transport Layer Security (TLS)**	You should use HTTPS everywhere, even for static sites. If you have an HTTP connection, change it to an HTTPS one.	X	X	1
Non-standard Authorization		**Use OAuth 2.0**	OAuth 2.0 has provided delegated authorization since 2012. OpenID Connect added federated identity on top of OAuth 2.0 in 2014. Together, they offer a standard spec you can write code against and have confidence that it will work across IdPs (Identity Providers). If you're communicating between microservices, you can use OAuth 2.0's client credentials flow to implement secure server-to-server communication	X	X	1
Hardcoded secrets	When you develop microservices that talk to authorization servers and other services, the microservices likely have secrets that they use for communication. These secrets might be an API key, or a client secret, or credentials for basic authentication. The #1 rule for secrets is don't check them into source control.	**Encrypt secrets at rest**	The first step to being more secure with secrets is to store them in environment variables. But this is only the beginning. You should do your best to encrypt your secrets.	X	X	1
Non-scalable security controls	Dependency and container scanning should be part of your source control monitoring system, but you should also perform tests when executing your CI (continuous integration) and CD (continuous delivery) pipelines.	**DevSecOps**	DevSecOps is the term many recommend instead of DevOps to emphasize the need to build security into DevOps initiatives.	X	X	1

	Non-proactive security	If someone tries to attack your APIs with hundreds of gigs of username/password combinations, it could take a while for them to authenticate successfully. If you can detect this attack and slow down your service, it's likely the attacker will go away. It's simply not worth their time.	**Use attack mitigation techniques**	You can implement rate-limiting in your code (often with an open-source library) or your API Gateway. I'm sure there are other options, but these will likely be the most straightforward to implement.	X	X	1
	Non-proactive security	The idea behind time-based security is that your system is never fully secure—someone will break in. Preventing intruders is only one part of securing a system; detection and reaction are essential, too.	**Use attack mitigation techniques**	Use multi-factor authentication to slow down intruders, but also to help detect when someone with elevated privilege authenticates into a critical server (which shouldn't happen that often). If you have something like a domain controller that controls network traffic, send an alert to your network administrator team whenever there's a successful login. This is just one example of trying to detect anomalies and react to them quickly.	X	X	1
60	**Publicly accessible microservices**		**Use API Gateway**	Pros of API Gateway: High security via SSL termination, Authentication, and Authorization. In large Corporations, API Gateway is compulsory to centralize security and cross-cutting concerns.	X	X	1
	Giant API Gateway	The Mobile client's API requirements are usually different from Web client as they have different screen size, display, performance, energy source, and network bandwidth.	**Use Backend for Frontend**	In a highly secured scenario where downstream Microservices are deployed in a DMZ network, the BFF's are used to provide higher security. Use BFF If an extra layer is needed between the UI and Downstream Microservices for Security reasons.	X	X	1

61	**Trust the Network**	Over time, trust-the-network has become an antipattern, and the industry is moving toward zero-trust networking principles.	**Zero Trust Network**	With zero-trust networking principles, you carry out security much closer to each resource in your network. Any microservices security design must take overall performance into consideration and must take precautions to address any drawbacks.	X	X	1
	Non-standard identity propagation	The challenge is to build trust between two microservices so that the receiving microservice accepts the user context passed from the other one. You need a way to verify that the user context passed among microservices isn't deliberately modified.	**Use JSON Web Tokens (JWT)**	Using a JSON Web Token (JWT) is one popular way to share user context among microservices	X	X	1
62	**Publicly accessible microservices**	The first step is to isolate and expose only the top-level services and completely restrict access to other services. We thus reduce the attack surface, thereby focusing on securing the few top-level services	**Use API Gateway**	For instance, only the top-level services are wired to an external load balancer or API gateway, while others are unreachable from outside the domain.	X	X	1
	Non-secure communication	Lack of transparency of cloud infrastructure prompts enterprises to deem securing communication between microservices critical.	**Transport Layer Security (TLS) mutual authentication**	One of the standard ways to do this is via MTLS (mutual transport layer security). Mutual TLS enforces both client and server to authenticate each other.	X	X	1
	Centralized authorization	top-level services need to follow security policies to safeguard content and delivery. However, enforcing policies requires standardized IAA (identification, authentication, and authorization).	**Use a token-based approach**	Standard techniques such as SAML, OAuth2, and OIDC allow authentication and authorization between clients and services. These techniques enable clients to obtain a token that encapsulates identification and authentication information.	X	X	1

63	**Decentralised authentication**	It is inconvenient that everyone might have to login separately for different systems, under a various usernames and passwords for each, and having complexity here by forcing a broker to do this job	**Centralise authentication**	Among the many possible ways to have a strong Authentication and Authorization, is the use of Single Sign On gateways because these can avoid the use of libraries that, despite they help to reduce duplicated code, rely in shared one. objective is centralized behaviors for redirecting the user and perform the handshake in only one place.	X	X	1
	Non-secure communication	In Microservices, services must communicate with each other in an implicit way and are exposed to a man-in-themiddle attack.	**Use transport Layer Security (TLS)**	It is advisable to use HTTPS instead of HTTP basic authentication not just due to the fact of encrypting user and password information. Using HTTPS guarantees that a given client is communicating with whom he wants to, providing additional protection against people eavesdropping on the traffic between client-server or messing with the payload.	X	X	1
	Non-encrypting sensitive data	Many breaches take place in protected environments, and information is attainable just because it is reachable at a given poorly secured point, as opposite areas of the system that well-guarded and consist in a costly attack target.	**encrypt data at rest**	its necessary to assure that data laying-around is contained in an encrypted way.	X	X	1

No layered security	Microservices acts in layer or cells, so designing a system that can act like an onion thus providing layer of security is essential.	**use defense-in-depth**	defense in depth is probably a last line of defense when the others are lingering or failing, so, the architecture of a given app should consider firewall over main layers of service controlling every port and service passing through it, recurring to deep packet inspection which is a combining technology of intrusion detection system and intrusion prevention systems with a stateful firewall	X	X	1
					IRR	91,90

Classification

	Integrity				Confidentiality			Authenticity		
	Own crypto code	Non-encryped data exposure	Hardcoded secrets	Non-secured service-to-service communication	Insufficient access control	Publicly accessible microservices	unnecessary privileges to microservices	Unauthenticated traffic	Centralized authorization	Multiple User Authentication
Paper Id	Use already validated encryption technologies	Encrypt all sensitive data at rest	Encrypt secrets at rest	Use mutual TLS	use OAuth 2.0	Add an API Gateway	follow the least privilege principle	use mutual TLS + use OpenID connect	use decentralized authorization	use single Sign-on
1	X				X	X				
2			X			X			X	X
3	X					X				X
4					X	X			X	
5		X			X	X				
6	X		X					X		
7		X		X			X	X		
8				X	X	X	X			
10				X				X		X
12				X	X					X
13		X		X	X					
15		X	X	X			X			
16				X		X				
17	X	X	X		X			X	X	X
18				X		X				X
19	X		X		X	X			X	
20	X	X		X			X			X
21				X			X			
22				X						
23				X		X	X			X
24				X	X	X	X			
25		X		X	X	X			X	X
26				X	X	X		X	X	
27					X	X	X	X		
28					X			X		
29		X		X						
30								X		
31								X		
32				X		X			X	X
33			X	X						
34							X			X

35				X			X			
36			X		X					
37		X		X	X			X		X
38					X	X		X		X
39									X	
40	X			X						
42						X				X
43					X	X			X	X
44				X					X	
45				X						
46				X	X	X			X	X
47				X	X					
48		X		X		X	X		X	X
49				X	X					
50	X	X							X	
51						X				
52					X				X	
53				X	X		X	X		
54									X	
55				X				X	X	X
56					X	X		X	X	X
58	X				X	X				
59			X	X	X					
60						X				
61									X	
62				X		X			X	
63		X		X						X
	9	12	8	32	25	25	12	14	19	20