Master's Degree Project Proposal

Distributed Information Flow Control in Serverless Computing

Prasannjeet Singh

Author: Prasannjeet Singh
Supervisor: Narges Khakpour
Course Code: 5DV50E
Semester: VT 2019
Department: Computer Science
In Collaboration with: Omegapoint AB
Company Supervisor: Daniel Deogun, VP Academy, Omegapoint AB

TABLE OF CONTENTS

1. BACKGROUND

In this document, we propose to construct an approach to secure a serverless system using *Distributed Information Flow Control*. For a system to be secure at the programming level, it is imperative that **confidentiality** and **integrity** is enforced. While confidentiality is described as the virtue of a system to protect it's sensitive data from being disclosed to unauthorized parties, where as integrity can be explained as the ability of a system to prevent an information from being manipulated or modified by unauthorized parties. Going forward, all the essential ideas will be elaborated in this section as follows: 1.1 introduces *Serverless Computing*, where as 1.2 talks about the information flow control model. Furthermore, 1.3 describes the security property of *non-interference* in security, and 1.4 describes two types of information flow control enforcements, *Termination Insensitive Non Interference* (TINI), and *Termination Sensitive Non Interference* (TSNI). Here, TSNI is described formally. Finally, 1.5 discusses existing approaches that employ IFC to secure serverless systems.

1.1 Serverless Computing

With the rise in usage of Cloud Computing, different architectures have been introduced by vendors over time, increasing efficiency, performance, and at the same time, decreasing the cost. In the early days, Infrastructure as a Service, or IaaS was prevalent, which provided virtual servers in the cloud. However they were still servers, with the difference being - the need to personally own the servers was not required anymore. Further, the concept of PaaS, or Platform as a Service was introduced, that delivered virtual application platforms, viz. JAVA Spring Boot, Python Flask, Node JS Express, etc. Here, the developers were no more responsible for maintaining the operating system. Nevertheless, packaging an application for deployment and managing the scalability was still critical. Although after the introduction of container orchestration platforms such as Docker Swarm, Kubernets, etc. PaaS has started gaining more popularity.

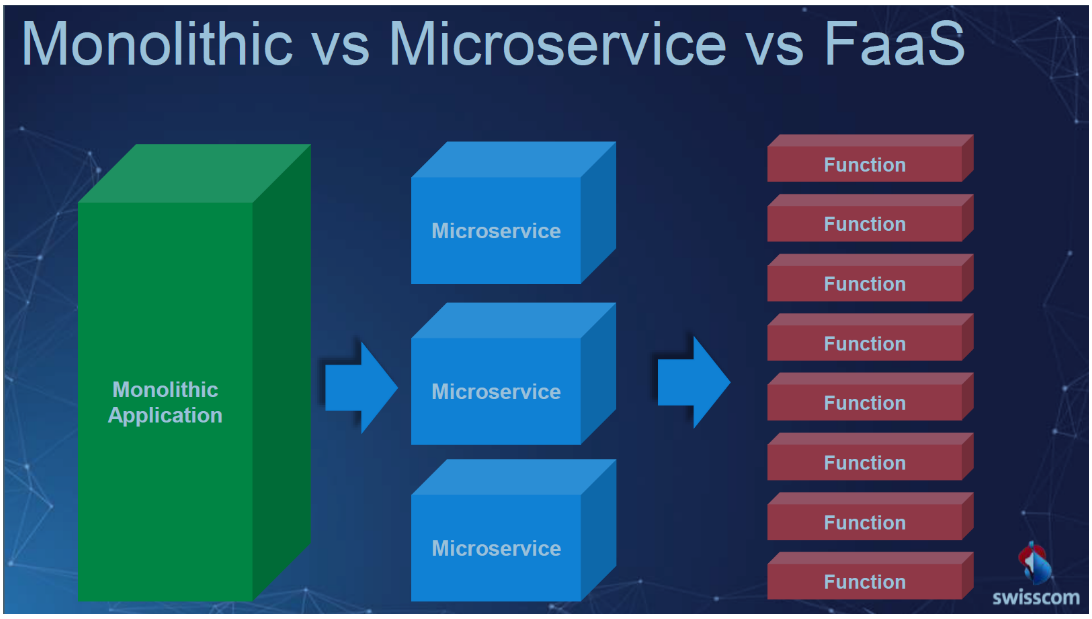

Fig. 1.1: Graphical representation of the architectural difference between a Monolithic Application (IaaS), a Microservices-Based Application (PaaS), and Serverless (FaaS)

Finally, Functions as a Service (FaaS), or *Serverless Computing* was introduced, which can intuitively be perceived as taking the resource abstraction to its logical conclusion [1]. It is needless to say that Serverless Computing still involves servers. However in this case, developers spend minimum time in maintaining or managing servers as compared to previous-generation architectures. Instead of accommodating a monolith application or a collection of microservices, Serverless architecture only contains a set of functions deployed individually. Figure 1.1 shows the contrast between the aforementioned architectures[2].

1.2 Information Flow Control

Information flow control, or IFC mainly focuses on building secure systems irrespective of the fact that the system might contain faults or bugs. It is difficult to build secure systems in the first place, as a single bug in any line of code may lead to security vulnerability. Therefore, it is not surprising to learn that in many cases, a simple bug has resulted in the disclosure of many sensitive information, such as personal identity, credit card numbers, etc.[3,4,5]. As a result, there are a lot of strategies available currently that focuses on finding and fixing bugs such as SQL Injection, Temporary file races, Buffer overflows, Missing access checks, Format string bugs, Integer overflow, etc. However, it is important to note that most of these bugs require different methodologies in order to tackle them; and therefore, it has eventually become a battle between the attacker and security expert for who find the next bug in a system first. In case of the former, the system might be exploited and in case of the latter, the bug might be fixed, and the system made secure – until the next bug is discovered. As it can be observed, this approach to make the system secure does not appear to be practical, as it can eventually turn out to be risky and costly, since declaring a system completely bug-free is not possible.[6,7,8]

Let us contemplate a small illustration to support our argument – consider a simple eCommerce website that sells books online. The web-application is built under different layers of abstraction, as shown in the Figure 1.2. The web application itself enforces some kind of security policy that restricts one user to access the credit-card or address information of any other user. Going a level down, the web server, such as Apache could have its own security policy that ensures which web browser can make what http requests, etc. Going deeper, the operating system might have some enforcements to protect different Windows/UNIX users from one another. In the same vein, the hardware could be providing some mechanism to protect the kernel from application code, etc. As we can intuit, for the whole system to be secure, the code at every abstraction layer must be correct, or bug-free. However, as we are aware, all these layers combined comprise of millions of lines worth of code, and therefore, occurrence of one or more bugs anywhere up the stack is inevitable. Consequently, it can be said that the strategy of securing a system by the means of fixing a bug is practically impossible. This implies that securing a system completely via fixing all the discovered bugs is impossible. Nevertheless, via Information Flow Control, it is indeed possible to build a secure system despite the existence of bugs in the code.

Let us now take a step back and realise the fact that ensuring the security of a system is, in varying degrees, dependent on the data, rather than the code. As an instance, in the example shown in above (Figure 1.2), a user wouldn't want their credit-card data to be unauthorizedly accessed or sent to another location by an attacker. Assuming we can *control and monitor all these data movements in a system*, and so long as the data remain secure, it is of no concern to supervise what is being done to the data by the application code. Trying to secure a system through this approach achieves the goal to secure the system without caring about the existing bugs in the system. Furthermore, enforcing IFC at lower abstraction level in an application ensures that the application remains secure in all the above abstraction levels. IFC is conceptually a simple idea: A label is associated with every piece of data in a system, and this label usually applies to data at all different levels of abstraction, ranging from *hardware* to the *application data*. Additionally, this

[1] S. Waterworth, "Introduction to Serverless Computing," 2018. https://www.instana.com/blog/introduction-to-serverless-computing/

[2] B. Christner, "Docker Serverless," 2017. https://pt.slideshare.net/BrianChristner/docker-serverless

[3] L. H. Newman, "The Equifax Breach Was Entirely Preventable," Wired, 2017. https://www.wired.com/story/equifax-breach-no-excuse

[4] Big data breach! Aadhaar software hack raises major security concerns, BusinessToday. https://www.businesstoday.in/current/economy-politics/aadhaar-software-hack-uidai-data-ghost-entries/story/282260.html

[5] T. Armerding, "The 18 biggest data breaches of the 21st century," CSO Online, 2018. https://www.csoonline.com/article/2130877/the-biggest-data-breaches-of-the-21st-century.html

[6] SteelKiwi Inc, "Is There Such a Thing As Bug-free Software?," Hackernoon, 2018. https://hackernoon.com/is-there-such-a-thing-as-bug-free-software-320cd862af17.

[7] R. Varshneya, "There's No Such Thing as a Bug-Free App," Entrepreneur India, 2015. https://www.entrepreneur.com/article/251742

[8] Allen, "The Myth of 'Bug Free' Software," BetaBreakers, 2016. https://www.betabreakers.com/the-myth-of-bug-free-software

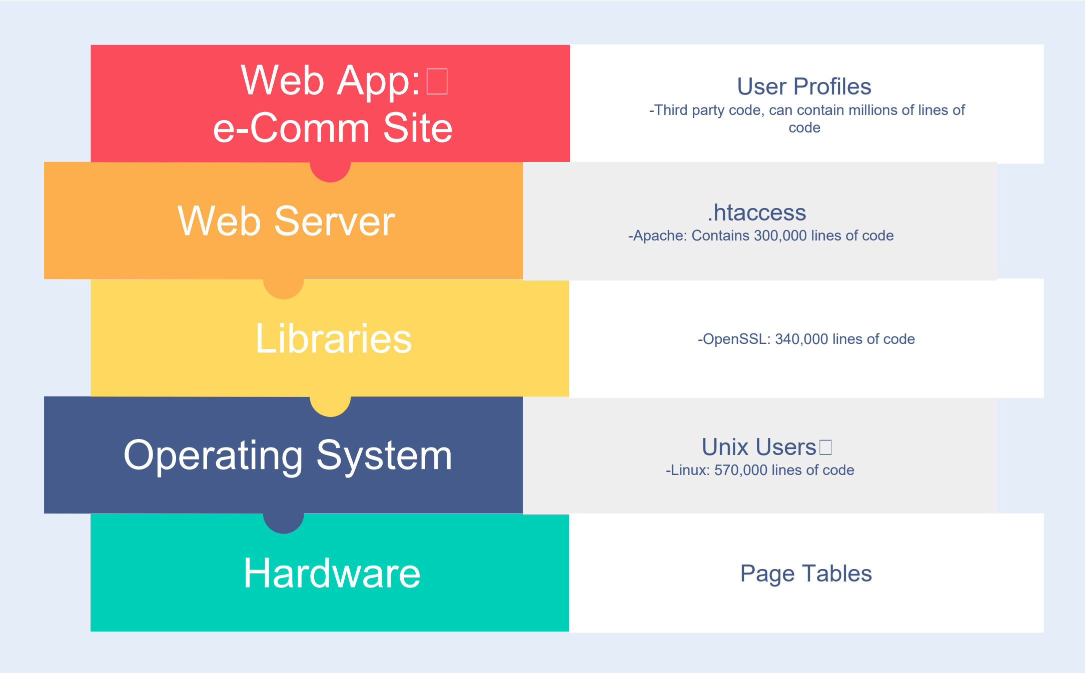
Web App:
e-Comm Site
User Profiles
-Third party code, can contain millions of lines of code
Web Server
.htaccess
-Apache: Contains 300,000 lines of code
Libraries
-OpenSSL: 340,000 lines of code
Operating System
Unix Users
-Linux: 570,000 lines of code
Hardware
Page Tables

Fig. 1.2: A software architecture diagram for a simple e-commerce web application.

label follows the data as it moves around through the system, and ultimately this label specifies what can happen to the data. For instance, it can make sure that the copy a user's credit-card number cannot be sent to an attacker's website. If this idea of Information Flow Control is implemented appropriately, the hope is that a system could be made secure without concerning ourselves with numerous existing bugs in the system.

The information flow model is an extension of the state machine model that constantly monitors the situation of a system to restrain it from entering an insecure state. A system that supports a state machine model must have the potential states of its process inspected in all the circumstances to confirm that they are controlled. Under the benchmarks laid down by the model, a system that boots up in a secure state and preserves the security of any transaction throughout itself shall, at all times, be in a secure state. Consequently, the information flow model consists of state transitions, objects and lattice states that regulate data flow policy. The fundamental purpose of information flow model is to restrict the flow of insecure and unauthorized data in any direction throughout the system. The exchange of data across different systems could be regulated using *guards*, that can be employed by the information flow model. *Flume* is a popular decentralized information flow control (DIFC)[1] model and system "*that applies at the granularity of operating system processes and standard OS abstractions (e.g., pipes and file descriptors)*"[2].

1.3 Non Interference

Non-Interference in this research refers to the protection of the secret (*high-sensitive*) part of the memory from an attacker who can observe the public (*low-sensitive*) part of the memory[3]. It is a policy that imposes an attacker to not be able to make a distinction between two computations from their outputs, in case they vary only in their secret inputs. It was first proposed in 1982[4], and is a further development of the information flow model built to assure that the objects and subjects at different security level do not interfere with each other. In this context, the subjects may be applications, system users, or processes, while objects can be processes, bits of data, programs or documents. Additionally, *information flow monitors* is an evolved technique that dynamically enforces noninterference. As one would expect, dynamic monitors enforce non interference differently for different instances of the execution of process under observation. This is because the information control flow of a program can be different for different input values, thus making the program secure for some set of values, while declaring it insecure for others. This is in contrast with static monitors, that will completely declare a program insecure, even if it becomes insecure for a very small range of input values. Therefore, it can be said that dynamic monitors allows analysis in further details, as compared to static monitors, as it is in the beginning of the execution of a dynamic monitor, when the assignment of security levels to the variables takes place. Moreover, the assigned security levels do not change throughout the execution. Adding to this, there also exists several *hybrid* monitors [5]–[7] that combine both static and dynamic methodologies.

It must, however be noted that even dynamic monitors are prone to produce false positives. Hence, there is a need to construct improved monitors that allow multiple *paths*. These challenges were scoped by Khakpour and Skandylas in their research [8] where a new approach was proposed that, at some designated checkpoints, supervises a program written in a subset of Java based on boolean supervisory controller synthesis [9] to construct a hybrid monitor that, to avoid future leakages, applies suitable countermeasures in checkpoints by forecasting security breaches. This approach is supported by a tool implemented in OCAML and uses SOOT to analyze the program at the java bytecode level.

1.4 Termination Sensitive Non Interference

Information Flow Control (IFC) [10], [11] based security techniques could be an optimistic approach to tackle the security problem in Serverless Applications. However, a security policy known as Termination Insensitive Non-Interference (TINI) have been implemented by most of the previous implementations. Intuitively, it is ensured by TINI that a program's non secret outputs are not allowed to expose any secrets stored in the system. On the other hand, the fact that the system ceased to generate outputs could be utilized to deduce some part of the secret. This channel, also known as the *termination-channel* is usually neglected as it just leaks one bit. Furthermore, Askarov et al. have claimed that it might leak more than just a bit [12]. However, for Serverless Applications, even the leakage of a single bit might be dangerous, as these are highly scalable and triggering parallel computations, each leaking one bit could be performed to access the secure data. Another strong security policy, called as Termination Sensitive Non Interference (TSNI) [13] could be enforced that eliminates the termination channel. Let us formally define TSNI followed by observing an example:

Consider a Program **P**:

```
input h from cH
while h>0 do skip
```

We define some notations as:

cH	Input Channel
P	The above program
μ	Memory
$State < P, \mu >$	A program state.
$terminating(P, \mu, I)$	If the $State < P, \mu >$ is executed with input I, then it terminates.
$I_1 =_L I_2$	both I_1 and I_2 belong to the same security level **L**, which can either be public or confidential.

$State < P, \mu >$ is TSNI $\forall I_1 \& I_2 \iff$

$$I_1 =_L I_2 \wedge terminating(P, \mu, I_1) \implies terminating(P, \mu, I_2) \tag{1.1}$$

TSNI either converges or diverges $\forall$ **I** $\equiv$ **L**

As an example, for the program **P** above, let us consider a **State** $< P, \mu >$ where two inputs I_1 & I_2 belong to the same channel **cH** and are of the same security level, which is *confidential*. Therefore, $I_1 =_L I_2$, where L is a *confidential* security level. Let us now consider the values of I_1 & I_2:

- $I_1 = [cH : 0]$
- $I_2 = [cH : 1]$

It is known that in program P, an input where h>0 will result in an infinite loop, whereas any input that does not satisfy the above condition will terminate. Therefore, based on the input values assigned to I_1 & I_2, we can say that:

$$I_1 =_L I_2 \wedge terminating(P, \mu, I_1) \implies \neg terminating(P, \mu, I_2) \tag{1.2}$$

As it can be clearly observed, the Equation 1.2 above contradicts the definition of TSNI in Equation 1.1 in Page 5. Hence, it can be stated that the **State** $< P, \mu >$ does not satisfy TSNI.

1.5 Existing Approaches

Google Scholar[9], which could be considered as one of the biggest search portal for scientific literature was used to look for other existing approaches that uses Information Flow Control to secure a Serverless system. Following were the set of keywords used:

Keywords Used

*(tsni **OR** tini **OR** ”termination sensitive non interference” **OR** ”termination insensitive non interference” **OR** ”information flow control” **OR** ”distributed information flow control” **OR** difc **OR** ifc) **AND** (serverless **OR** ”serverless computing”)*

[9] Google Scholar - https://scholar.google.co.in/

Based on the above keyword-combination, we could only find one relevant research that uses Information Flow Control to secure a Serverless Computing system. This approach, proposed by Alpernas et al. [14] aims to secure a serverless computing system by ensuring confidentiality via enforcing termination sensitive non interference (TSNI). The tool developed by the authors, called as Trapeze Framework [10] is written in JavaScript and at the moment, only supports the node.js runtime. The author claims that TSNI enforcement is essential in case of serverless computing, since the termination channel, which usually leaks just a bit cannot be disregarded; as the attacker can amplify the termination channel by executing many parallel computations, each leaking one bit.

[10] Trapeze Framework - https://github.com/kalevalp/trapeze

2. MOTIVATION

The functions deployed in Serverless run on-demand and webmasters are billed by the millisecond of execution time. Therefore, if the function is unused, it's *free*. In this case, everything that the application needs to run is completely abstracted by the cloud platform, and in turn, it handles failover, scalability, load balancing, etc. Like other architectures, the server-side logic is still written by the application developers but in contrast, the logic here is in the form of individual functions that are run in event-triggered and ephemeral stateless compute containers and are managed by a third party completely. The containers are created on demand and destroyed as soon as the function is executed. These functionalities have catered to Serverless architecture gaining huge popularity over time (Figure 2.1)[1]. Amazon's AWS Lambda [2] is currently the most popular provider of Serverless Architectures. However, other players like Google Cloud Functions [3], Azure Functions [4], IMB Cloud Functions [5] and Pivotal Function Service [6] have also started gaining popularity.

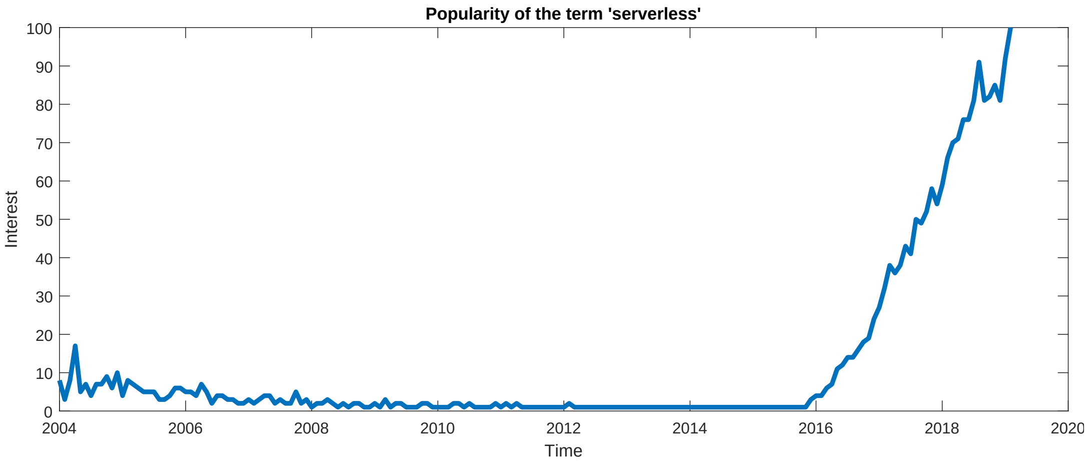

Fig. 2.1: A trends graph from Google showing the search interest of the term *serverless* over the past fifteen years. Numbers represent search interest relative to the highest point on the chart for the given region and time. A value of 100 is the peak popularity for the term. A value of 50 means that the term is half as popular.

With the benefit of pay-per-use, zero-administration, ease of deployment and auto-scaling, Serverless also comes with new vulnerabilities. It does not have adequate security testing, as testing a serverless application can be far more complex than standard applications. As a result, many scanning tools haven't yet adapted to Serverless applications. Secondly, the application has an increased attack surface, as it consumes data from many event sources, namely cloud storage, HTTP APIs, IoT device communications, message queues, etc. A report from Puresec [15] describes in detail the ten most critical security risks in Serverless architectures. Hence, a robust solution for the security of Serverless applications is needed.

[1] Google Trends, "Popularity of the keyword Serverless," 2019. http://bit.ly/GoogleServerless

[2] Amazon, "AWS Lambda – Serverless Compute - Amazon Web Services." https://aws.amazon.com/lambda/

[3] Google, "Cloud Functions - Event-driven Serverless Computing." https://cloud.google.com/functions/

[4] Microsoft, "Azure Functions—Serverless Architecture." https://azure.microsoft.com/en-us/services/functions/

[5] IBM, "IBM Cloud Functions." https://console.bluemix.net/openwhisk/

[6] Pivotal Software, "Pivotal Function Service (PFS)," jan 2017. https://pivotal.io/platform/pivotal-function-service

3. PROBLEM STATEMENT

Being a fledgling technology, Serverless is yet to receive the required attention, and as a result, it is strongly vulnerable to external attacks. Furthermore, it is important to ensure Confidentiality and Integrity of data in an application. The virtue of a system to allow only authorized users the access to protected and sensitive data while shielding the data from harmful invaders is called Confidentiality. Additionally, Integrity alludes to the methodologies that ensure the authenticity of data, and the fact that it is accurate, and confidential.

IFC models have previously been implemented for traditional applications. The proposed work by Khakpour et. al [8] works on Symbolic Confidentiality Guards Synthesizer, or SCGS,[1] which is a static analysis tool that primarily identifies any confidential data leak automatically in object-oriented programs. The aforementioned work supports centralized IFC where it synthesizes security guards for single methods in an application. In this work, we will extend it with decentralized IFC to enforce confidentiality and integrity in serverless computing.

Moreover, an IFC system for Serverless Applications has also been developed [14] as discussed previously. While the former enforces TINI for traditional JAVA applications, the latter implements the TSNI model for Serverless functions written in JavaScript for the node.js runtime. Furthermore, it only focuses on data *confidentiality* and does not enforce data *integrity*. Therefore a strong framework is needed for the security of Serverless Applications that enforces both data *confidentiality* and *integrity*. Additionally, other programming languages should also be focused upon. Based on the above discussions, we can elicit the ensuing **research questions** targeted to our study:

RQ1: Can an approach designed for native Java applications to enforce confidentiality and integrity be equally effective for a Java based serverless application? Why?

RQ2: What is the recommended approach to make a Java based serverless application secure, using distributed information flow control?

RQ3: Can a tool that analyzes the Java based serverless application be built based on the aforementioned approach?

[1] N. Berthire and N. Khakpour, "Symbolic Confidentiality Guards Synthesizer — (SCGS), " Department of Computer Science, University of Liverpool and Department of Computer Science and Media Technology, Linnéuniversitetet, 2018. https://cgi.csc.liv.ac.uk/~nberth/scgs/..

4. CONTRIBUTIONS

In continuation to what was discussed in the Problem Statement, an IFC modeled tool in the form of an Artifact that enforces TSNI will be fabricated that will, either statically or dynamically monitor the information flow of a Serverless JAVA program, and ascertain its security. It will be an extension to [8] that enforces TINI and was made for native JAVA applications. Therefore, the key contributions for this thesis would be the following:

1. A distributed IFC approach will be proposed for enforcing security in serverless computing that will extend the current centralized approach[8].

2. Intra-behavioral analysis will be considered to check and enforce security in serverless computing.

3. The approach will be supported by a tool built upon the current tool[8].

4. The approach will be tuned to the nature of serverless computing.

5. APPROACH

To investigate and find a solution for the problem described above, an approach of **Design Science Research (DSR)** [16] will be followed, wherein a new artifact, in the form of a solution to the aforementioned problem will be created that will include generating new knowledge. Simon describes *artifact* as an entity created by humans, rather than something occurring naturally [17], and they can include algorithms, system design methodologies, human/computer interfaces or development processes[1]. DSR is defined as the methodology to fabricate a meaningful artifact to address a previously determined, and meaningful problem[18]. The DSR process for this research will be composed of four mutually dependent, yet methodologically distinct steps [19]:

- **Problem Analysis**: Here, the problem will be thoroughly analyzed by performing a systematic literature review in all the related aspects of security in Serverless computing. Current methodologies will also be examined carefully and the possibility to apply their ideas in this DSR will be investigated. It will also consist of learning previously developed tools and performing test experiments that will be used in this thesis, like SCGS, discussed in Chapter 3.
- **Artifact Design - 1** This step will consist of proposing an informal high-level model of DIFC in serverless computing by extending SCGS approach, i.e. which informal security policies should be enforced, who and where should they be applied.
- **Artifact Design - 2** In this step, we will be formalizing the proposed DIFC framework.
- **Artifact Construction** With a clear roadmap in mind after the problem analysis and artifact design, the next step will be to create the artifact that can provide confidentiality and integrity in a Serverless application, i.e. we will implement the framework and extend SCGS, if needed.
- **Artifact Evaluation** After successful construction of the artifact, it will be tested against many Serverless applications. Furthermore, vulnerabilities will also be injected manually into a program to check the performance of the artifact. We will also evaluate DIFC from effectiveness and performance point of view by applying it on a few case studies.
- **Interpretation, theory construction and learning** Finally, the research will be articulated as per the steps taken throughout the process.

[1] J. Hagelback, Design Science - Degree Projects in Computer Science, 2019. https://coursepress.lnu.se/subject/thesis-projects/design-science/

6. TIME PLAN

Date	Milestone	Deliverables
11th February, 2019	Planning for the degree project finished	Submission of the first draft proposal
1st March, 2019	Revision of the draft proposal	Submission of the final project proposal for the degree project
4th March, 2019	Literature review on IFC, TSNI, TINI, Non-Interference, Serverless Security	Completing the background section of thesis and gathering enough information to start working on the artifact.
20th March, 2019	Extensive working with Soot, ReaTK, SCGS to learn the working environment	
20th April, 2019	Designing the artifact	Answering the research question
10th May, 2019	Implementing the artifact	Answering the research question
15th May, 2019	Analyzing and testing the artifact	Artifact
30th May, 2019	Submission of the improved and approved thesis report	Thesis Report
5th June	Preparing the Presentation	Presenting the Thesis

Tab. 6.1: The proposed time plan for degree project

7. COOPERATION FORM

For this degree project, I will be meeting my Supervisor bi-weekly, or weekly. I will be sending my progress report via e-mail at regular intervals as well.

I will also interact with my Company Supervisor via email, or via meeting personally based on their availability.

BIBLIOGRAPHY

[1] A. C. Myers and B. Liskov, “Protecting privacy using the decentralized label model”, *ACM Trans. Softw. Eng. Methodol.*, vol. 9, no. 4, pp. 410–442, Oct. 2000, ISSN: 1049-331X. DOI: 10.1145/363516.363526. [Online]. Available: http://doi.acm.org/10.1145/363516.363526.

[2] M. Krohn, A. Yip, M. Brodsky, N. Cliffer, M. F. Kaashoek, E. Kohler, and R. Morris, “Information flow control for standard os abstractions”, in *Proceedings of Twenty-first ACM SIGOPS Symposium on Operating Systems Principles*, ser. SOSP ’07, Stevenson, Washington, USA: ACM, 2007, pp. 321–334, ISBN: 978-1-59593-591-5. DOI: 10.1145/1294261.1294293. [Online]. Available: http://doi.acm.org/10.1145/1294261.1294293.

[3] A. Sabelfeld and D. Sands, “Dimensions and principles of declassification”, in *18th IEEE Computer Security Foundations Workshop (CSFW’05)*, Jun. 2005, pp. 255–269. DOI: 10.1109/CSFW.2005.15.

[4] J. A. Goguen and J. Meseguer, “Security policies and security models”, in *1982 IEEE Symposium on Security and Privacy*, Apr. 1982, pp. 11–11. DOI: 10.1109/SP.1982.10014.

[5] P. Shroff, S. Smith, and M. Thober, “Dynamic dependency monitoring to secure information flow”, in *20th IEEE Computer Security Foundations Symposium (CSF’07)*, Jul. 2007, pp. 203–217. DOI: 10.1109/CSF.2007.20.

[6] A. Russo and A. Sabelfeld, “Dynamic vs. static flow-sensitive security analysis”, in *2010 23rd IEEE Computer Security Foundations Symposium*, Jul. 2010, pp. 186–199. DOI: 10.1109/CSF.2010.20.

[7] G. Le Guernic, A. Banerjee, T. Jensen, and D. A. Schmidt, “Automata-based confidentiality monitoring”, in *Advances in Computer Science - ASIAN 2006. Secure Software and Related Issues*, M. Okada and I. Satoh, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 75–89, ISBN: 978-3-540-77505-8.

[8] N. Khakpour and C. Skandylas, “Synthesis of a permissive security monitor”, in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 11098 LNCS, Springer, Cham, Sep. 2018, pp. 48–65, ISBN: 9783319990729. DOI: 10.1007/978-3-319-99073-6_3.

[9] N. Berthier and H. Marchand, “Discrete controller synthesis for infinite state systems with reax”, *IFAC Proceedings Volumes*, vol. 47, no. 2, pp. 46–53, 2014, 12th IFAC International Workshop on Discrete Event Systems (2014), ISSN: 1474-6670. DOI: https://doi.org/10.3182/20140514-3-FR-4046.00099. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1474667015373791.

[10] D. E. Denning and D. E., “A lattice model of secure information flow”, *Communications of the ACM*, vol. 19, no. 5, pp. 236–243, May 1976. DOI: 10.1145/360051.360056.

[11] D. Hedin, D. Hedin, and A. Sabelfeld, “A Perspective on Information-Flow Control”, 2011. DOI: 10.1.1.295.9015.

[12] A. Askarov, S. Hunt, A. Sabelfeld, and D. Sands, “Termination-insensitive noninterference leaks more than just a bit”, in *Computer Security - ESORICS 2008*, S. Jajodia and J. Lopez, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 333–348, ISBN: 978-3-540-88313-5. DOI: 10.1007/978-3-540-88313-5_22.

[13] A. Sabelfeld and D. Sands, “A Per Model of Secure Information Flow in Sequential Programs”, *Higher-Order and Symbolic Computation*, vol. 14, no. 1, pp. 59–91, 2001. DOI: 10.1023/A:1011553200337.

[14] K. Alpernas, C. Flanagan, S. Fouladi, L. Ryzhyk, M. Sagiv, T. Schmitz, and K. Winstein, “Secure Serverless Computing Using Dynamic Information Flow Control”, Feb. 2018. arXiv: 1802.08984.

[15] O. Z. S. S. A. Segal, “The Ten Most Critical Security Risks in Serverless Architectures”, Puresec, Tech. Rep., 2018.

[16] S. T. March and G. F. Smith, “Design and natural science research on information technology”, *Decision Support Systems*, vol. 15, no. 4, pp. 251–266, Dec. 1995. DOI: 10.1016/0167-9236(94)00041-2.

[17] H. A. Simon, *The sciences of the artificial*. MIT Press, 1996, p. 231, ISBN: 0262193744.

[18] J. F. Nunamaker, M. Chen, and T. D. Purdin, “Systems Development in Information Systems Research”, *Journal of Management Information Systems*, vol. 7, no. 3, pp. 89–106, Dec. 1990. DOI: 10.1080/07421222.1990.11517898.

[19] T. Mettler, M. Eurich, and R. Winter, “On the Use of Experiments in Design Science Research: A Proposition of an Evaluation Framework”, *Communications of the Association for Information Systems*, vol. 34, no. 1, Jan. 2014. DOI: 10.17705/1CAIS.03410.