Protecting Secrets in Cloud Applications using Moving-Target Defense

Development of a moving-target defense solution for mitigating cross-VM attacks in the cloud

Master's thesis in Computer science and engineering

ERIK VAN BENNEKUM
FELIX SCHULZE

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2023

Master's thesis 2023

Protecting Secrets in Cloud Applications using Moving-Target Defense

Development of a moving-target defense solution for mitigating cross-VM attacks in the cloud

ERIK VAN BENNEKUM
FELIX SCHULZE

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2023

Protecting Secrets in Cloud Applications using Moving-Target Defense
Development of a moving-target defense solution for mitigating cross-VM attacks in the cloud
Erik van Bennekum
Felix Schulze

Supervisor: Ahmed Hassan, Department of Computer Science and Engineering
Examiner: Risat Pathan, Department of Computer Science and Engineering

Master's Thesis 2023
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: A nimble hare with aviator goggles darting across a meadow, evading a fox, which is falling behind in the tall grass.

Typeset in LaTeX
Gothenburg, Sweden 2023

Protecting Secrets in Cloud Applications using Moving-Target Defense
Development of a moving-target defense solution for mitigating cross-VM attacks in the cloud
Erik van Bennekum
Felix Schulze
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

Over the last decade, more and more IT systems are moved from on-premise or co-located servers to cloud infrastructure to take advantage of the reduced cost, complexity and time-to-market that cloud infrastructure brings. However, a shared environment, such as a server shared between different customers, exposes customers to sophisticated side-channel attacks, where a malicious virtual machine can steal information from any of the other virtual machines running on the same host. This thesis proposes a solution to this problem by utilizing moving-target defense, where the virtual machine of the customer is moved to different physical machines on a regular basis to avoid any adversary from having enough time to perform long-running side-channel attacks. To solve the connectivity problem, where clients need to connect to this moving virtual machine, a reverse proxy is used that keeps track of the current location of the virtual machine and keeps the connections alive. Benchmarks show that the added latency is insignificant for most applications, and the slight reduction in throughput is unlikely to become a bottleneck.

Keywords: moving-target defense, mtd, security, cybersecurity, cloud, proxy, virtual machine, vm, side-channel attack

Acknowledgements

We would like to thank our supervisor, Ahmed Ali-Eldin Hassan for his support and valuable feedback during the project, as well as our examiner, Risat Pathan, for his feedback and good questions. We would also like to thank Chalmers University for providing us with the tools that enabled us to complete this project.

Erik van Bennekum, Felix Schulze, Gothenburg, 2023-08-15

Contents

List of Figures

List of Tables

1

Introduction

Infrastructure-as-a-service (IaaS) as provided by vendors such as Amazon Web Services (AWS), Google Cloud Platform (GCP) and Microsoft Azure bring many benefits over hosting your own infrastructure, such as flexibility and economies of scale [1]. A common way of deploying an application to the cloud is to provision a Virtual Machine (VM) to a physical machine shared by multiple customers, thereby reducing the upfront and operational expenses compared to buying and managing your own physical on-premise IT infrastructure. But with such advantages, the cloud also exposes us to new threats. One such threat is a side-channel attack of an adversary running a VM on the same physical machine, a so-called cross-VM side-channel attack.

While the provider will manage the physical security and software updates to managed services, the customer is still responsible for the security within their application. Although this reduces the effort compared to on-premise applications, additional precautions have to be taken in a shared setting such as a multi-tenant cloud environment for virtual machines. While the virtual machines are virtually isolated from each other, they still utilise the same hardware. This shared usage leaves some state physically in the hardware that can leak information by exploiting side-channel attacks [2]. As it takes a nonzero amount of time to successfully execute a side-channel attack, the attack needs to be disrupted in time to prevent it from succeeding.

Common configurations of software systems are relatively static, whether it is in the cloud or on-premise. Examples of static configuration parameters include names, addresses, software stacks, network configuration and physical location of deployment. The static nature of such configurations makes it easier for adversaries to map out and attack IT infrastructure, as they can take as much time as they need. Ideally, systems should be secure and capable to handle any attack, but it is not realistic to patch every conceivable security flaw before it can be exploited. A way to mitigate attacks is to change the configuration parameters of software systems over time to complicate and frustrate an attack. Changing the physical or network location of a running application, changing the routing of two communicating services or swapping one software implementation for another are all examples of Moving-Target Defense (MTD) strategies [3][4].

1.1 Goal

The goal of this thesis is to develop a moving-target defense system for cloud customers, with a minimal amount of static components to mitigate cross-VM side-channel attacks by denying any adversary sufficient time to successfully execute such an attack. We do this by moving the victim VM to a new host on a regular basis. In order for this solution to be viable it needs to be performant and reliable, which will be discussed in sections 1.1.1 and 1.1.2.

1.1.1 Performance

To be useful, the proposed solution should not be a bottleneck in a system, regardless of where and how it is deployed. Due to the dynamic nature of an MTD system, additional networking operations will occur. To limit the overhead of these operations, no processing should be done beyond traversing the network stack. To get a better understanding of the performance impact, we will benchmark the solution.

1.1.2 Reliability

Depending on implementation, MTD strategies may have reliability issues due to downtime while a service is being migrated from one host to another. The solution should consider the requirements of different applications, with regard to up-time and statefulness, as there is a trade-off between these two factors. The precise requirements depend on the application and are further discussed in section 3.1.

1.2 Scope

The moving-target defense solution will be aimed at mitigating co-location attacks, any other attack vectors will not be taken into account. Furthermore, seeing that the migration of a VM from one machine to another per definition mitigates co-location attacks, we will not test the effectiveness of the MTD solution. Because network applications use TCP extensively across the board, the MTD system will only support TCP for simplicity. Since the goal of this MTD solution is to mitigate threat vectors in the cloud, testing of the MTD solution will be done on AWS, which simulates real-world deployment. This thesis supplies the concept of a novel MTD solution as well as a proof-of-concept implementation of said solution. It is outside the scope of this thesis to produce an implementation ready for production use, and may only serve as a starting point for an independent security solution.

1.3 Contributions

Our contribution is a new, platform-agnostic moving-target defense solution which provides protection against co-location attacks without negatively affecting the cost or complexity of a given system. The first part of the solution is the novel way of placing the components to solve common issues with moving-target defense can

be a basis for further research and practical implementations. The second part is the theory regarding the adjustments made in the system for different types of applications using the moving-target defense solution.

2

Background

In this chapter we discuss the technical concepts on which the work is based. we give a brief explanation of virtual machines and side-channel attacks, as well as the threat model and how they relate to each other. The concept of a reverse proxy is discussed, which is what this MTD solution largely relies on. Finally, we discuss related work and its nuances.

2.1 Virtual Machines

The virtual machine, as originally defined by Popek and Goldberg[5], is an implementation of computer machinery in software, rather than hardware. Virtual machines come in two general variants, the process virtual machine and the system virtual machine. The first provides a run-time for an application by providing an abstraction for a high-level programming language, such as Java running on the Java Virtual Machine. System virtual machines emulate an entire physical machine and all its hardware components, with the goal of allowing multiple operating systems and its applications to run concurrently on one physical machine in isolation. Only the system virtual machines are relevant for this thesis and will be referred to as virtual machines.

The physical machine running one or more virtual machines is called the host, and each virtual machine running is a guest on the host. In an environment with multiple virtual machines on one host, the CPU cores can be shared among multiple guests. This could mean that one physical core is assigned to different guest VMs in different points in time. Guest VMs leave state on the physical core, such as cache lines. When this physical core is then assigned to a different VM, it exposes this state left by the first VM, giving rise to a side channel.

A common way to get access to computing power without your own infrastructure is to rent a VM through a cloud provider, such as AWS, with their Elastic Compute Cloud (EC2) offering.

2.2 Side-channel attacks

Side-channel attacks are a type of attack vector where physical attributes are used to tap into a system [6]. There are three general categories of side-channel at-

tacks: time-driven, trace-driven and access-driven. Time-driven attacks operate on repeated observation of clocks to estimate the total duration of operations that a victim performs, such as encryption and decryption operations, and using that information to learn information about secret keys.
Trace-driven side-channel attacks utilise traces leaked by the operation into the environment, for example, power usage, electromagnetic radiation or acoustics of a hard-disk spinning on a disk read.
Access-driven attacks utilise access to the same physical hardware or the ability to run a program on the same physical machine as the target victim. Such an attack monitors and manipulates the state of architectural components shared between the attacker and the victim. Examples of components that can be used in these attacks are caches, branch prediction tables, floating-point units, and shared components between concurrently executing threads on a single core in a simultaneous multi-threading machine [7].

2.3 Threat model

The threat model for this paper is an access-driven side-channel attack by an adversary having full control of a co-resident VM running on the same physical machine as the victim VM, as is common in cloud environments. The VM hypervisor, the software managing VMs on a physical host, is trusted and controlled by the cloud provider. As is standard in cloud environments, simultaneous multi-threading and memory deduplication are disabled, as there are known vulnerabilities associated with these features[8][9]. Within these circumstances, well-known attacks such as the one shown by Zhang et al [2] rely on shared per-core state, the ability to preempt the victim VM, and access to a system clock of sufficient resolution to steal secrets within a few hours.

2.4 Reverse proxy

A reverse proxy server, often referred to as just "proxy" for simplicity, is a type of program that exists in front of one or more services, and could be described as a friendly man-in-the-middle, as it relays traffic between a client and the services. This can be seen in Figure 2.1. A reverse proxy can inspect and transform the incoming data, as well as deciding where to send the data. The reverse proxy is usually deployed on the same network as the service(s) it proxies.

Common use-cases for a proxy are protecting a service from the internet, making multiple servers available through a single IP address, or adding functionality such as load-balancing or caching.

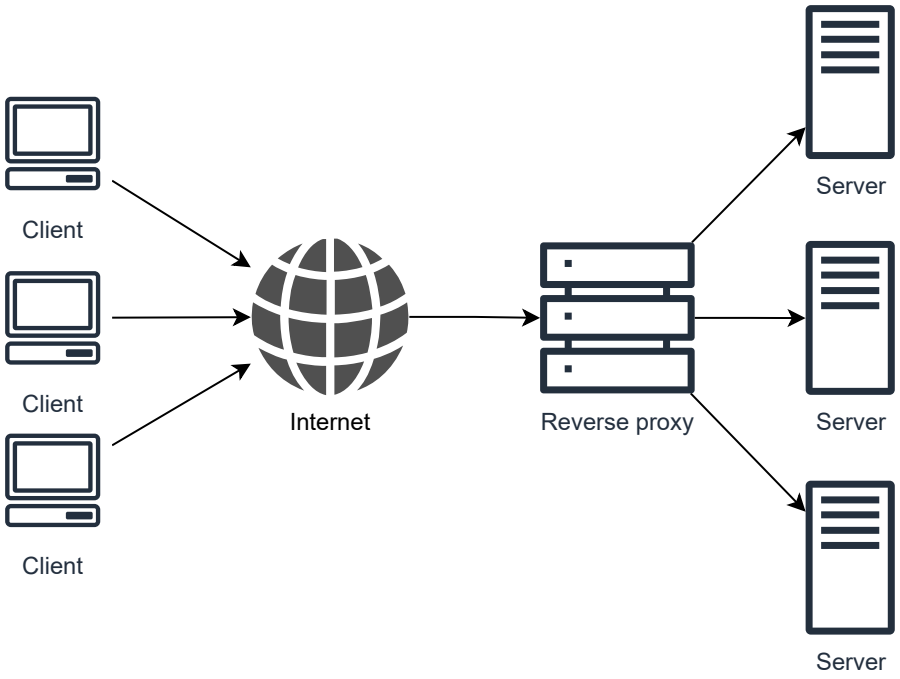

Figure 2.1: Example of a reverse proxy forwarding traffic from the internet to the appropriate server.

2.5 Related work

Using evasive maneuvres to avoid an adversary is a strategy that has been used for millennia by humans and animals alike. It is harder for any attacker to hit a moving target compared to one that is static. Moving-target defense, as used in the field of computer security, can come in different forms. One possible strategy utilising moving-target defense, as described by Buck & Tibom [10], is to have multiple containers running different software implementations performing the same function, such as different web-servers. These different containers rotate IP addresses, as to make it harder for any adversary on the network to send malicious data that is crafted for a specific software version.

A more well-know use of MTD is frequency-hopping, or frequency-hopping spread spectrum (FHSS), a mechanism used in various radio technologies that prevents interference and eavesdropping [11]. The carrier frequency of a transmission is rapidly changed in a way that is predictable for both sender and receiver, which greatly reduces the impact of interference, whether the interference is natural or a jamming signal. One of the many radio technologies that use FHSS is Bluetooth [12], therefore making it very likely that the reader has already used moving-target defense before.

There are several other MTD strategies and implementations, Ahmed and Bhargava [13] have proposed a framework for MTD in distributed systems; Al-Shaer et al. [14] and Caroll et al. [15] focus on MTD strategies that involve randomising network addresses of hosts; and Torkura et al. [16] proposed MTD mechanisms to overcome

the security implications of homogeneous microservices.

The differentiating factor between the work presented in this thesis and the mentioned previous work lies in the practicality of the deployment. Using a proxy, as described in section 3.2, makes it possible to deploy this MTD solution in any cloud-like environment that has an API to manage instances, such as AWS, Azure, GCP, Proxmox, Kubernetes, etc. It is even possible to have a mixed environment with multiple cloud solutions, as long as the MTD manager is able to reach all cloud APIs and proxies from where it is deployed, and the proxies can reach all services they proxy. Unlike previous implementations, our solution is able to maintain TCP sessions during migration. Additionally, there is no reliance on a load balancer, or hypervisor-specific management, to route traffic to the correct instance, making the integration of this MTD solution seamless.

3

Design & implementation

In order to evade the threats outlined in Section 2.3, the thesis will detail a moving-target defence solution with a focus on ease of deployment, utilising two components, a reverse proxy and a manager as depicted in Figure 3.1. An attacker needs a malicious VM to run for a significant amount of time on the same physical machine as the target VM to perform a cross-VM side-channel attack successfully. To avoid giving the attacker this time, the victim VM containing secrets needs to be moved to a different physical machine on a regular basis. The MTD manager performs this task by interacting with the cloud provider's API.

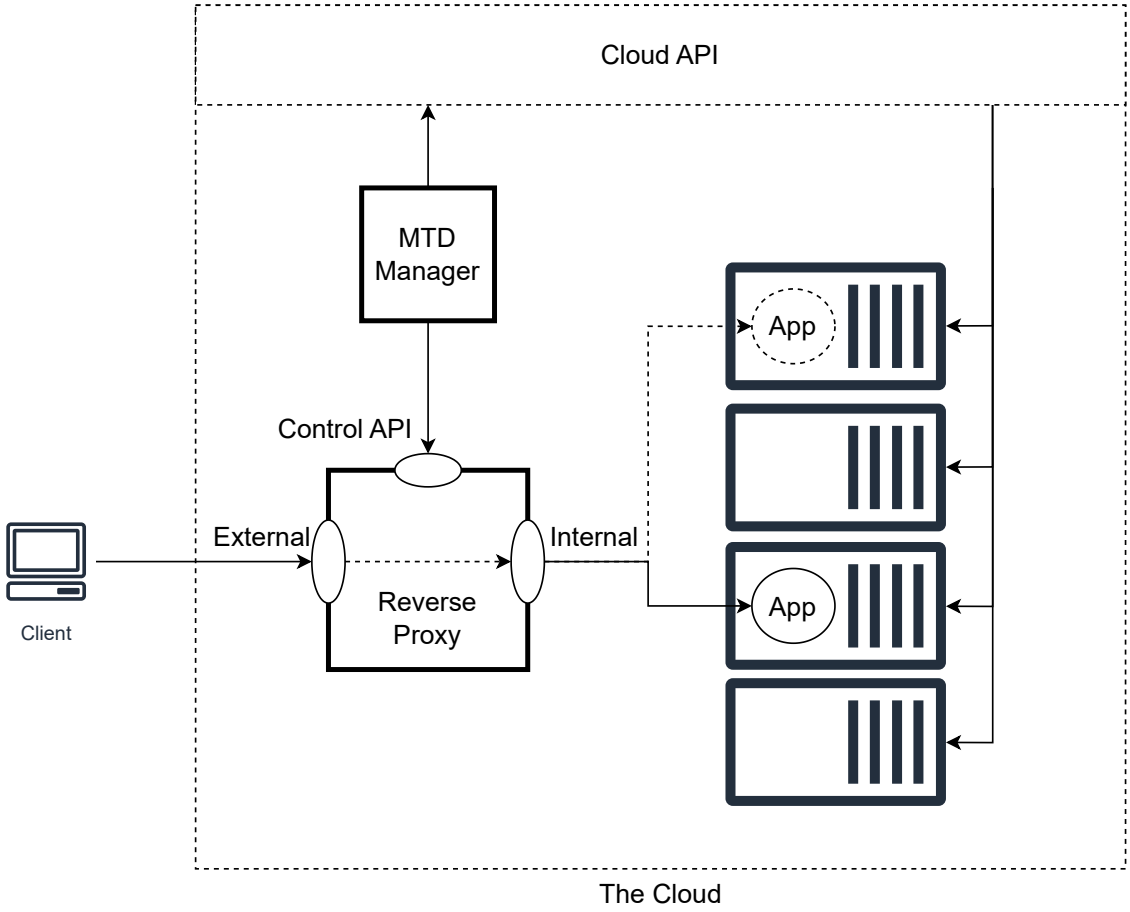

Figure 3.1: Architectural design of reverse proxy and MTD manager

We have chosen AWS as the cloud provider for demonstrating the moving-target defense technique, as AWS is the cloud provider with the largest market share [17]. AWS provides a suitable API and SDK available in multiple programming languages [18] that can be used to control cloud components with software. However, note that while the implementation of the MTD manager in this thesis contains AWS-specific parts for the creation and migration of VMs, it is entirely possible to change the software in a minimal way to make it suitable for other cloud providers or even on-premise installations.

A naive MTD solution may simply migrate services every so often. However, at the start of the migration, any client connected to the service would have its connection broken, and without any way of knowing where the service has moved to, there is no way for the client to reconnect to the service. This problem of knowing where to connect to after a migration of the service, could be solved by adapting existing software to communicate with the manager about the location of the service, dynamically re-configuring the application. However, a much simpler solution would be to put a reverse proxy in front of the server, which would eliminate the need to integrate the MTD manager with every possible service. In this case, the reverse proxy is what gets dynamically re-configured. An additional benefit of a reverse proxy is that the external connection between the client and the proxy can stay alive during the migration of the application. When the MTD manager has moved the service, it can inform the reverse proxy of the new location, after which the proxy can establish a new internal connection to the updated location. From the client's perspective, there might be a slight disruption during migration, depending on the migration strategy.

3.1 Migration strategies

Migration of the service between physical hosts is done by creating a machine image from a VM instance running the service, and creating a new VM from this image. This process is not instantaneous and depends on the hypervisor the virtual machines are running on. Depending on the type of application and its requirements, there are different migration strategies possible. We have implemented two different strategies, strategy A, suitable for applications that cannot afford to lose some state during migration, and strategy B, for applications that can handle some state loss. These strategies are described in subsection 3.1.1 and subsection 3.1.2, as well as being represented in Figure 3.2.

3.1.1 Migration strategy A - Consistency

The first strategy will stop the VM before creating an image, and therefore prevent normal operation of the service during migration. Operation is continued as soon as the new VM is successfully started from the image. No client will be able to get any response during this period, and any data sent to the server while the new VM is not ready is lost. This strategy makes sure that no state on the server is lost, as no state can be added to the old VM instance after the image has been created.

One example where this would be desirable is a web-shop database. Here it is better that a user is unable to place an order at all, rather than a user being able to place an order and get it confirmed, but this all happens while the database is being migrated and the order therefore disappears after migration, despite the user having been charged.

3.1.2 Migration strategy B - Availability

The second strategy will create an image from the VM without stopping the VM. The old VM will only be terminated after the switchover to the new VM has been completed. While the migration is happening, the old service is still able to answer any request coming from the clients, avoiding a gap in up-time that migration strategy A suffers from. While clients will not experience any downtime of the server, they might receive unexpected responses from the new server after the switchover, if the application relies on state that was sent to the old server during migration. For applications that are stateless or are able to afford to lose state during the migration period, and applications that require high up-time, this would be the preferred method.

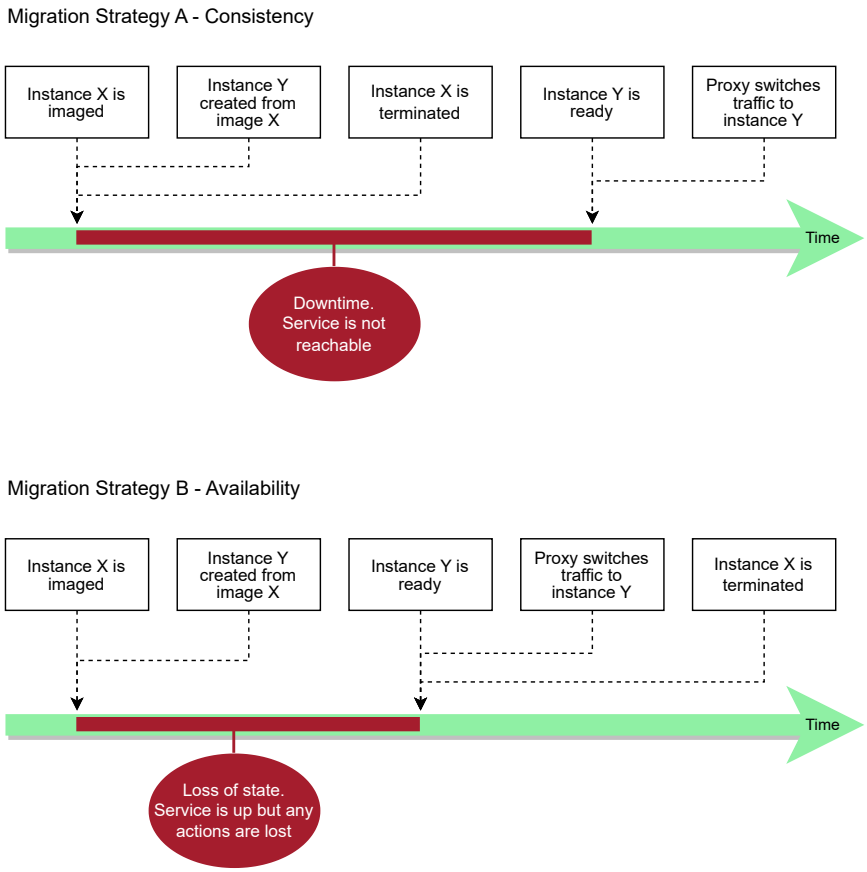

Figure 3.2: Migrations strategies depicted as executed by the MTD manager and their relative impact.

One example where this would be desirable is a web server serving secret company

documents. There is no downside to using this strategy since there is no application state that can be lost, and the migration will cause no practical downtime.

3.2 Run-time configurable reverse TCP proxy

The run-time configurable reverse TCP proxy, hereafter referred to as just "the proxy", behaves similarly to a software-defined network switch. This proxy does not move, contains no secrets of any kind and receives instructions from a manager through its control API. This manager is described in section 3.3.

The reverse proxy, a binary deployed in the cloud environment, as depicted in Figure 3.1, will be able to run several proxy tunnels. Multiple applications running the MTD strategy at the same time can use the same proxy, where each application needs one tunnel per exposed port. Each proxy tunnel consists of a TCP listener on a unique port on the external interface and a destination socket address where the associated service can be reached. When a client connects to the TCP listener and creates a new TCP connection to the proxy, the proxy will create a new connection to the service associated with the tunnel used. Any bytes received from the connection with the client will be sent over to the service over the associated connection, and any bytes received from the service will be passed on to the client. Multiple clients can connect through a single tunnel to a service.

The destination of a tunnel can be changed during run-time, causing any new client connections to be tunnelled to a different service instance. Existing client connections to the proxy stay alive, while the related connection between the proxy and the service gets replaced with a connection to the new service. When a tunnel is no longer desired, it can be dismantled, meaning that the proxy will terminate all the client connections and service connections belonging to that specific tunnel.

It is important to note that the proxy is a transparent component, meaning the data flowing through the tunnel is not modified by the proxy. Any encryption scheme, such as TLS [19] between the service and the client should function as normal and is not terminated by the proxy.

Table 3.1: Control API commands

Command	Arguments	Purpose
Create	tunnel ID, external port, internal port/IP	To create a new tunnel
Modify	tunnel ID, internal port/IP	To modify an existing tunnel
Delete	tunnel ID	To delete an existing tunnel

On starting the proxy, it exposes an HTTP endpoint for issuing commands to create, modify and terminate tunnels as given in Table 3.1. The MTD manager can use a control API at this endpoint to start a new tunnel, which will create a listening TCP socket at the specified port on the external interface. This allows a client to reach services through the proxy. Besides being able to create one or more tunnels, the control API can also modify existing tunnels, allowing the MTD manager to update the socket address of a service on the proxy after migrating it.

The implementation of the reverse proxy is critical for the performance of the network connections between the clients and the services. For the tunnels to work, data must traverse the network stack in the proxy, in addition to any potential processing done to the data. To minimise this overhead, and reduce any negative performance impact that the MTD solution has on a system, the raw data is proxied without being processed.

Considering that the proxy is one of the two components in the system that are static, simplicity and robustness is of utmost importance. The main attack vector of the reverse proxy is the control API, as it could be used to redirect traffic to a malicious service controlled by an adversary. Therefore, it is important that the control API is only reachable from the internal network of the cloud environment, not from the internet. As an additional security measure, the reverse proxy can check any incoming commands for a provided signature using a public key, confirming authenticity and integrity.

Choosing a programming language has a significant impact on the performance properties of the implementation, and choosing a language based on the relative speed would result in the choice of the C programming language [20]. However, according to the National Security Agency, programming languages that are not memory safe are not to be used anymore [21], as a large portion of security vulnerabilities in large C and C++ code bases are memory safety problems [22]. This leaves The Rust programming language as our next choice as a fast [20] and memory-safe [21] programming language.

Since the reverse proxy is a networking application without any CPU-intensive processing, it is IO bound. This type of application is most suited for asynchronous programming, and the most widely used asynchronous framework within the Rust ecosystem is the Tokio framework [23]. Within a Tokio application, there is the concept of tasks, which are asynchronous units of work. Each task runs concurrently, and if the run-time is configured to do so, tasks can also run in parallel, taking advantage of modern multi-core processors.

Table 3.2: Control API endpoints

URI endpoint	HTTP method	Description
'/'	GET	Get general information
'/command'	POST	Supply command in JSON format

The asynchronous tasks that are running within the proxy are given in Figure 3.3. The *command task* is listening for commands from the manager to create, modify or delete a tunnel. We do this by starting an HTTP server and exposing the endpoints in Table 3.2. After the manager has sent a command using the control API, the *command task* will spawn a new *tunnel task*, in case a new tunnel needs to be created, or updated, a shared variable that indicates the tunnel state is used. A newly spawned *tunnel task* starts a new tunnel by creating a TCP listener on the external interface, to which new clients can connect. If the shared variable changes, it will either delete the TCP listener, if the tunnel is to be terminated, or ignore

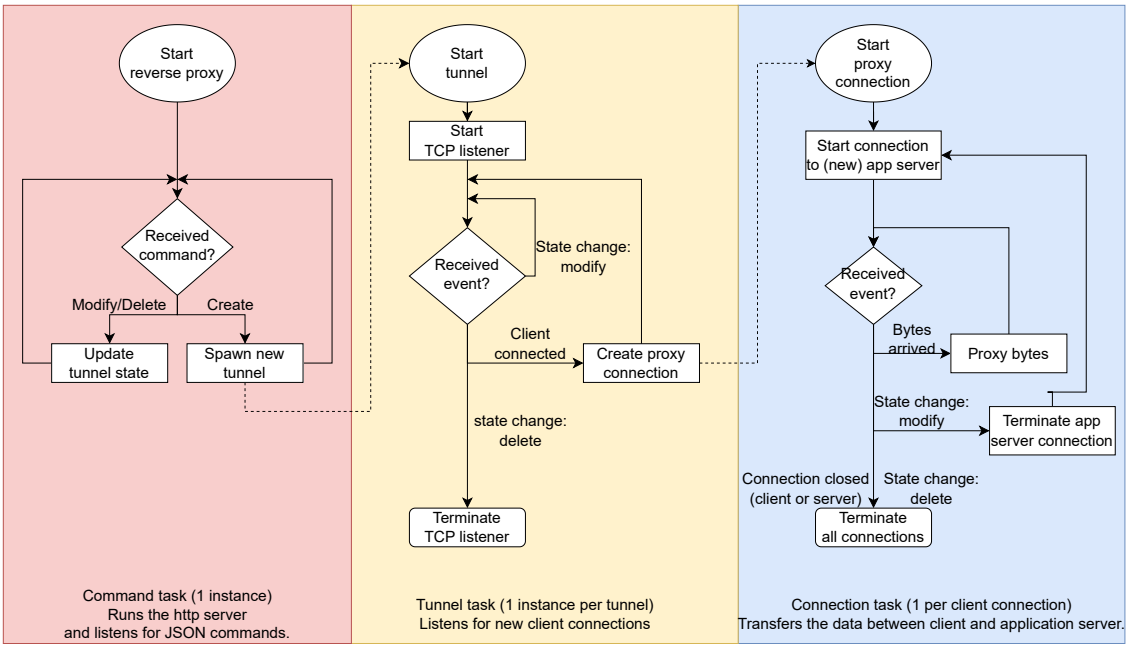

Figure 3.3: Tasks and control flow of the reverse proxy.

the change. When a client connects, it will spawn a new *connection task*. The new *tunnel task* will create a connection to the service and will forward any data that is received from the client or service to the other. When a change in the shared variable is detected, and the tunnel is modified, the *tunnel task* will replace the connection from the proxy to the service with a connection to the new service. In the case that the tunnel is terminated, both the connection to the client and to the service are closed.

3.3 MTD manager

The MTD manager has the responsibility to execute the migration strategies by moving the services and updating the proxy tunnel accordingly. It runs as a static component inaccessible from the internet. For every service that it needs to manage, it runs a timer to indicate when it is time to move the VM. On expiry of this timer, the virtual machine image is migrated to a different physical host. Depending on the migration strategy, the procedure for migration is slightly different. As shown in Figure 3.2, migration strategy A, used for consistency, the old VM is stopped at the same time that the image is created. For migration strategy B, aimed at availability, the old VM is terminated as soon as the switchover happened. To create an image from a VM, to start a new VM from the created image, and to terminate the old VM, the cloud API is used. To perform the switchover, but also to create or destroy a tunnel, the manager sends commands as given in Table 3.1 to the endpoint exposed by the proxy.

For management of AWS VM's, the manager uses the AWS SDK [18] and can therefore be implemented using any of the programming languages for which the

SDK is available. However, as it is desirable to avoid run-time errors as much as possible, a compiled language with static type checking is preferable. The choice was made to use the Go programming language because of its flexibility, speed and simplicity, as any cloud providers other than AWS are yet to be implemented and it should therefore be easy to understand and expand the code of the MTD manager. Go is a compiled, statically typed, and garbage-collected programming language. It is designed by Google for network and cloud applications, and to replace their existing services written in Python and C++, and is commonly used as "glue" between different API's and systems.

4

Tests & Benchmarks

To analyse the functionality of the system, we run several functional tests and performance benchmarks. The functional tests are there to prove that the system functions according to the specifications. Since there are several places where there is overhead compared to a traditional server setup in the cloud, performance benchmarks are done to determine the measure of performance degradation by utilising this moving-target defence strategy.

4.1 Functional requirements

The goal of the moving-target defence system is to resume operation after migrating the service to a new physical host without disconnecting the client from the proxy. Depending on the migration strategy, there are additional requirements. Migration strategy A adds the requirement that no state is lost during the migration period, at the expense of some downtime of the server. Migration strategy B has the requirement that the client sees no downtime during the migration period. However, using migration strategy B, the client could see some lost state. We will showcase the functionality by demonstrating several use cases.

To test a real-world use-case, a monitoring stack is deployed, consisting of three applications. The three applications are *Prometheus' Node Exporter*, *Prometheus* and *Grafana*. Each of these applications will be installed as a docker container in separate VM instances on AWS, to make this test easy to reproduce.

Node Exporter is a stateless application reading machine metrics from the host machine and exposing an HTTP endpoint from where those metrics can be collected, thereby exporting the metrics of the node it is running on, explaining its name.

Prometheus is a stateful monitoring application for collecting metrics from different sources. It will fetch the metrics produced by Node Exporter and store the time-series data.

Grafana is a stateless, customisable, visualisation dashboard application. It fetches and visualises the data collected by *Prometheus*.

Two tests will be done on this stack. For the first test, the moving-target defense will be applied to the *Node Exporter*, while the other components remain static, as shown in Figure 4.1. For the second test, the *Prometheus* application will be

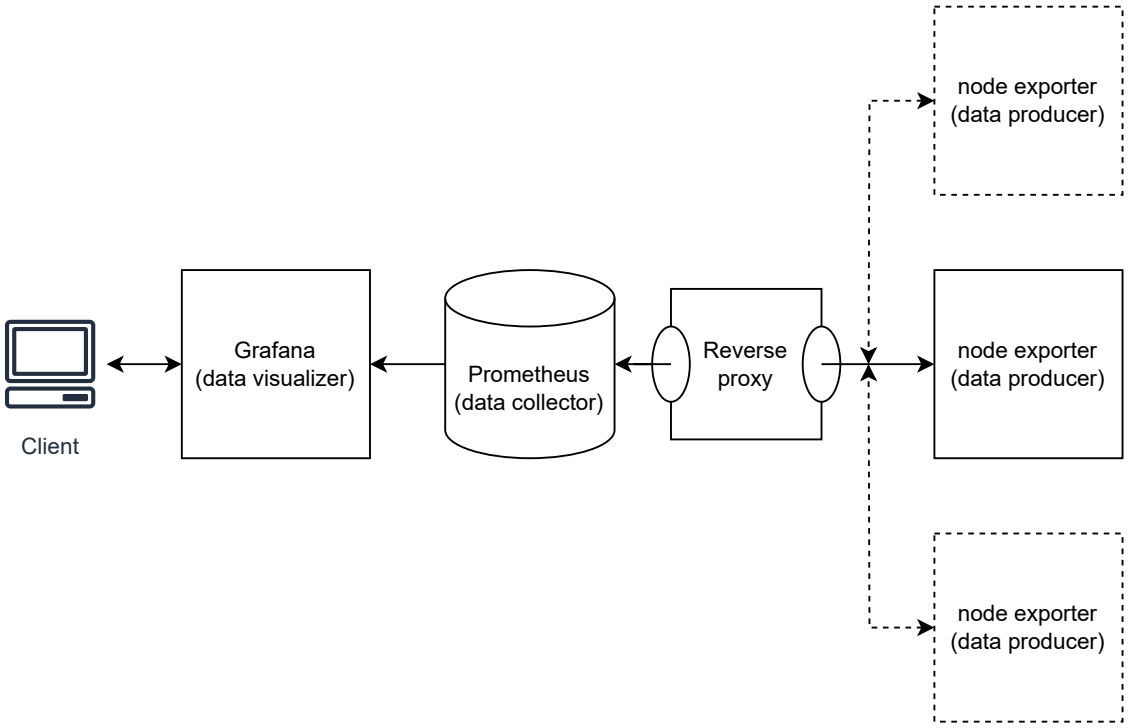

Figure 4.1: Functional test setup with monitoring stack moving the stateless data producer

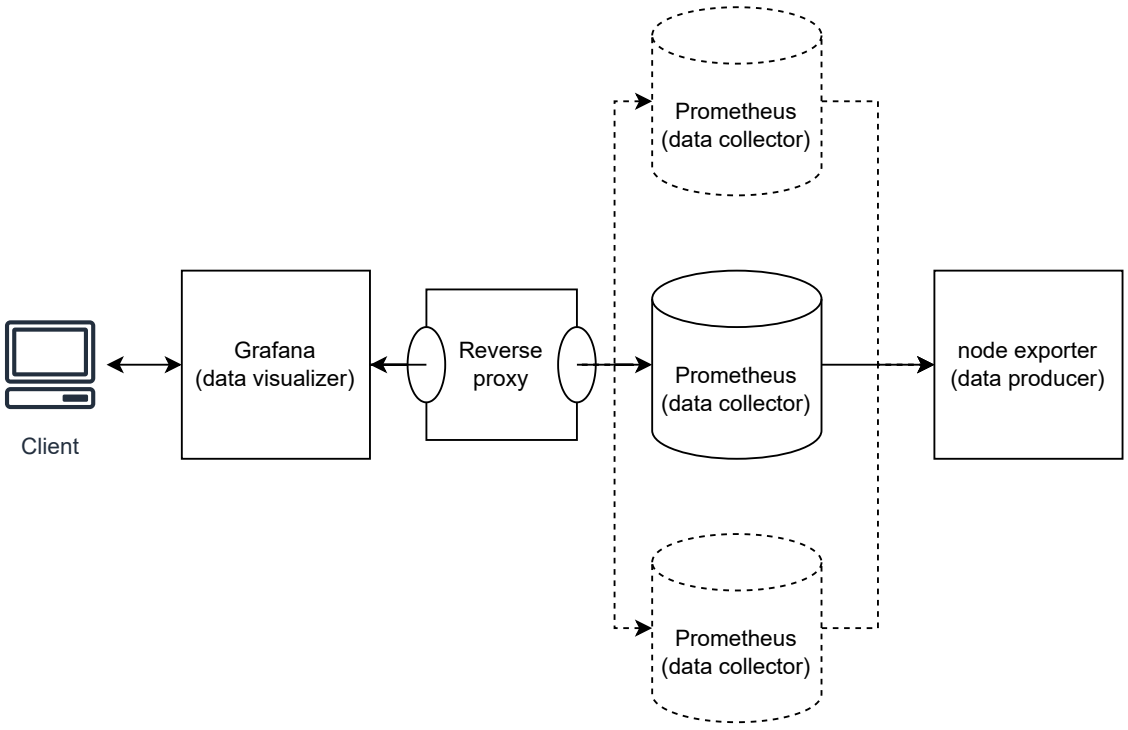

Figure 4.2: Functional test setup with monitoring stack moving the stateful data collector

moved around, as shown in Figure 4.2. For all tests, both migration strategy A and migration strategy B will be applied.

4.2 Benchmarks

The MTD manager is not benchmarked, as its performance does not affect the system and the system could even operate without it entirely, although this would void the functionality of the moving-target defence. The reverse proxy will be benchmarked for throughput, latency and reliability, as it is undesirable for the proxy to be a bottleneck in a cloud environment, limiting its practicality and usability.

The tests will be done both with the client, server, and proxy on the same physical machine, as well as in different virtual machines within the same availability zone on AWS. The first allows for measuring the pure overhead of the proxy on the application and kernel level, while the AWS test will mimic real-world usage better without being affected by unreliable external network connections. All AWS tests are run on a mix of t2.micro and t2.medium EC2 instances, which are VMs with 1 vCPU and 1GB of RAM, and 2 vCPUs and 4GB of RAM respectively.

The throughput of the proxy is measured by using the *iperf* [24] tool to measure throughput between two hosts with and without having the reverse proxy in the middle. The difference between the two numbers gives an indication of how much the reverse proxy is limiting the throughput. The VMs used for iperf test have a 5Gbps network interface card. It is of interest to note the total bandwidth of many simultaneous clients, compared to a single client, to give an indication of the scalability of the proxy. To simulate many simultaneous clients, the *iperf* tool supports parallel connections. The throughput will be measured with a single client, 5, 10, 50 and 100 clients. Each test will be run for 1 minute.

The round-trip time, a strong indication of latency, is measured using a custom-built benchmarking tool. Just like *iperf*, the tool is written with a server component and a client component. The server component consists of a simple TCP echo server, which will echo any bytes received on a connected socket. The client will measure the difference in timestamp between the moment it sends bytes to the server, and the moment it receives those bytes back. 50000 TCP packets are sent for the case where there is a proxy between the server and the client, and the same number for the case without a proxy in the middle. Between sending packets, the client waits 10 milliseconds to avoid congesting the network, as high network utilisation might increase latency.

In order to validate reliability two metrics will be used, uptime and packet delay variation (PDV), also known as jitter. The packet delay variation is quantified as the standard deviation of the round-trip time measured with the custom tool. The uptime is a function of the downtime during migration, and the time between migrations, as given by Equation 4.1. The downtime depends on the platform used but will be measured for AWS EC2.

$$uptime = \frac{\text{migration period} - \text{downtime}}{\text{migration period}} \tag{4.1}$$

5

Results

This chapter will provide the results of the functional test and the benchmarks to validate the moving-target defence strategy and the implementation of the reverse proxy.

5.1 Functional achievement

The MTD-manager was able to run for hours without interruption and was able to provide consistent and very frequent moving-target defence migrations.

Figure 5.1 shows the uptime of the Node Exporter under migration strategy B, which is aimed at consistency. The downtime marked in red is due to the gap between the new VM being ready (which is when traffic is switched to it), and the application on the VM being ready. Unfortunately, migration strategy A was not tested, as the toggle between migration strategies was not implemented in time.

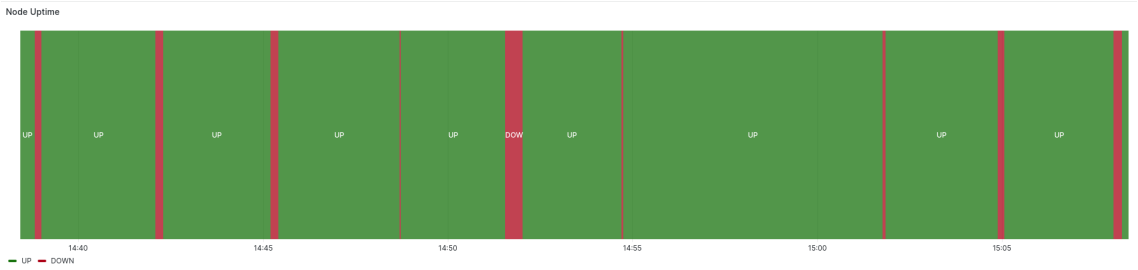

Figure 5.1: Uptime of data producer Node Exporter, as reported by Prometheus and graphed in Grafana. Data producer migrated using migration strategy B - availability.

When using migration strategy B, aimed at uptime, any data added to the old VM, after migration is started, will be lost. This can be seen in Figure 5.2, where the top half of the figure shows a screenshot of an arbitrary metric collected on the old VM running the application collector right before handover, while the bottom half shows the data collected on the new VM right after handover. The data collected during migration is missing from the bottom part of the figure, indicating state loss during migration.

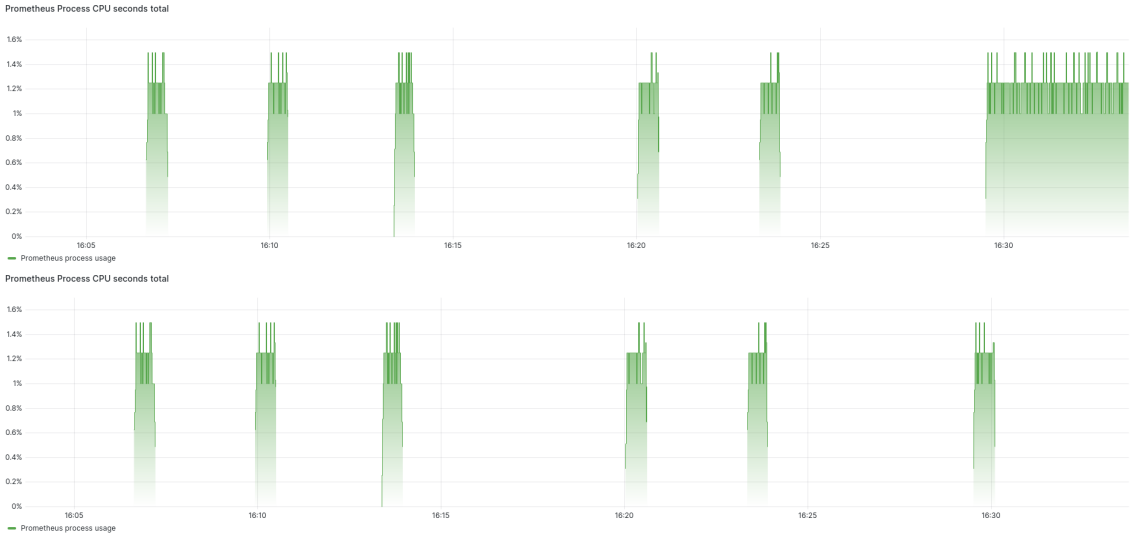

Figure 5.2: A demonstration of data integrity when using Migration strategy B on the data collector Prometheus. At timestamp 16.30 the new image is made, which means data recorded after that time disappears once the migration is complete.

5.2 Performance benchmarks

The first benchmark metric is the round-trip time between the client and the server. The round-trip time is measured with the client, proxy and server on the same physical machine and with the client, proxy and server on different VMs in AWS. For both situations, 50 thousand round trips are measured, of which the first 1000 are shown in Figure 5.3. As given in Figure 5.4, the average round-trip time is higher when there is a proxy present, both for the local test and the test in AWS. It is noticeable that the median round-trip time on a single machine with proxy is almost the same as the median of the round-trip time on different VMs in AWS, but that the standard deviation is significantly higher when the packets need to go over the network in the AWS test.

The absolute change in round-trip time measured by adding a proxy is given in Figure 5.5. The average increase in round-trip time for the local test is slightly more than 250 microseconds and for the AWS test around 700 microseconds. Percentage-wise, this is an increase of around 75% for both tests, as given in Figure 5.6.

The standard deviation of the round-trip times can be compared to the average round-trip time to get an indication of connection stability. Figure 5.7 shows the standard deviation as a percentage of the average round-trip time.

When running iPerf over the network on AWS, the results were close to the theoretical limits, but at higher amounts of parallel connections the proxy started choking, as can be seen in Figure 5.8.

When running iPerf locally on a AWS VM, up and down its own network stack only, the results showed a larger difference between the default configuration and running with the proxy enabled, as can be seen in Figure 5.9.

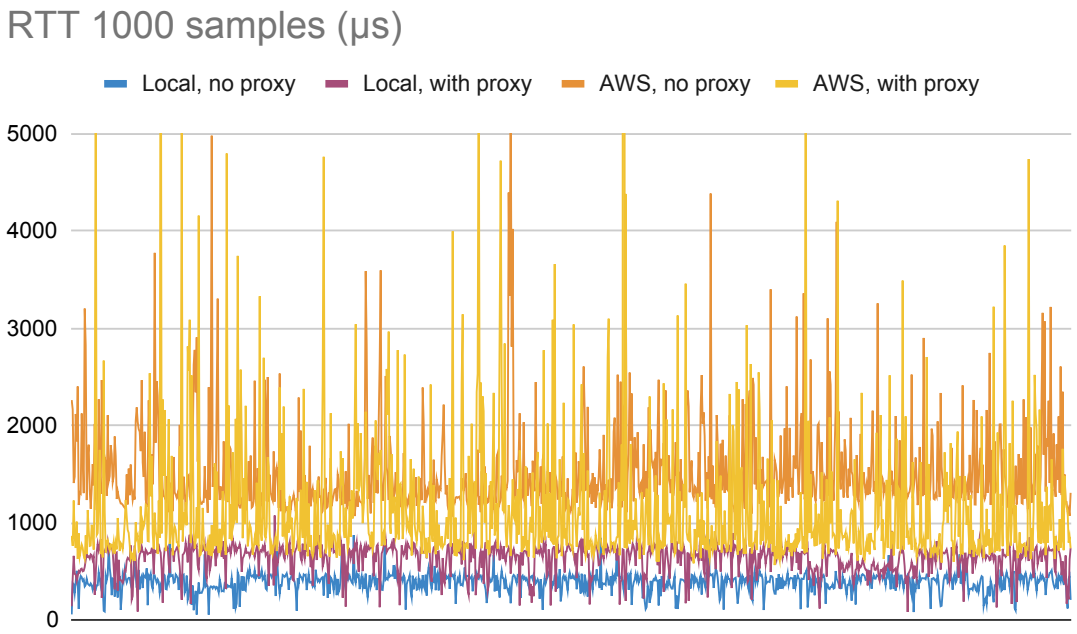

Figure 5.3: The first 1000 samples of round-trip times in microseconds. A smaller difference between "no proxy" and "with proxy" is better.

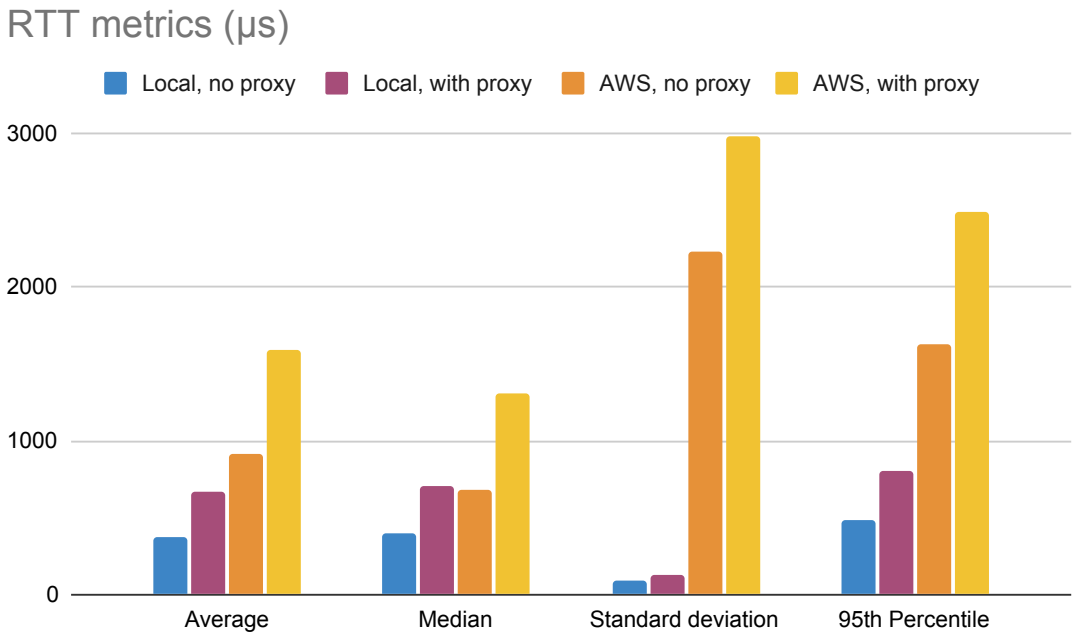

Figure 5.4: Different latency metrics in microseconds. A smaller difference between "no proxy" and "with proxy" is better.

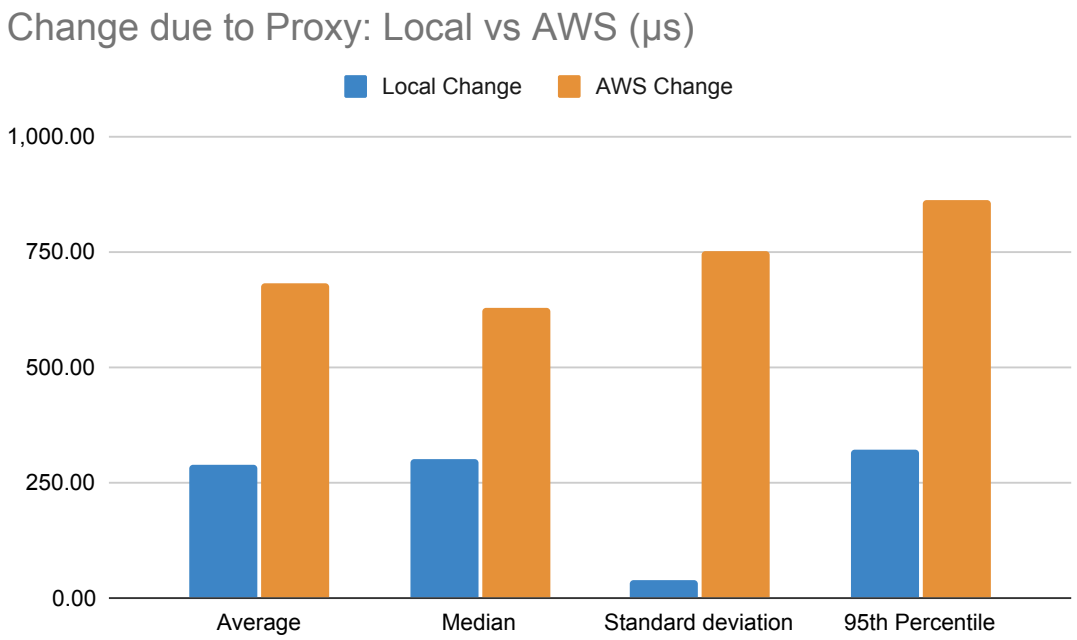

Figure 5.5: The impact of the proxy to different latency metrics, measured in microseconds. Positive values show an increase in latency when the proxy is active. Lower values are better.

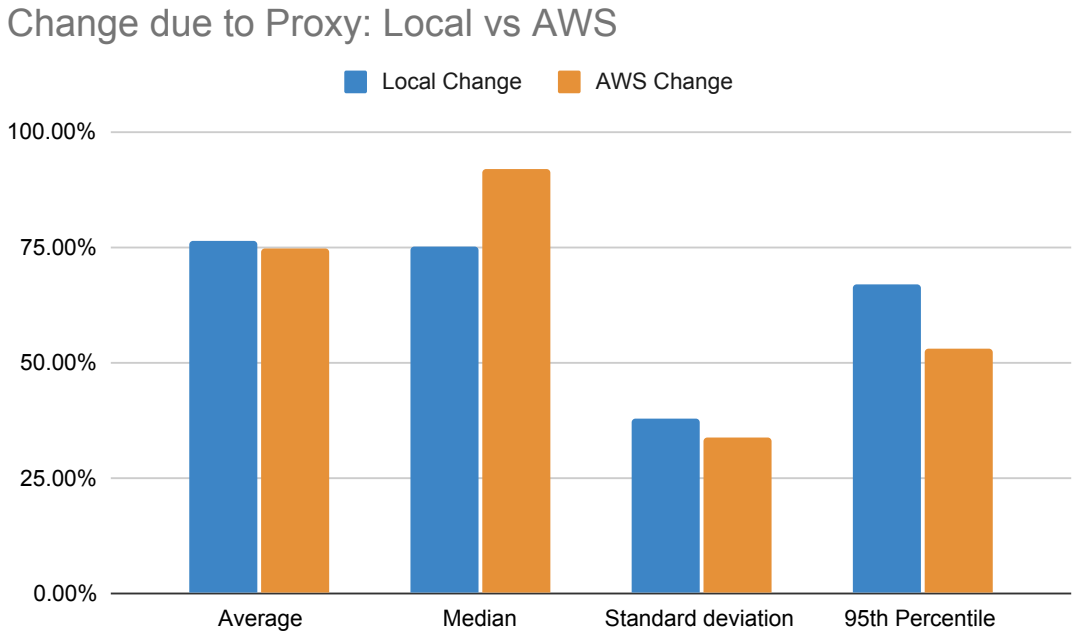

Figure 5.6: The impact of the proxy to different latency metrics, measured as a percentage. Positive values show an increase in latency when the proxy is active. Lower values are better.

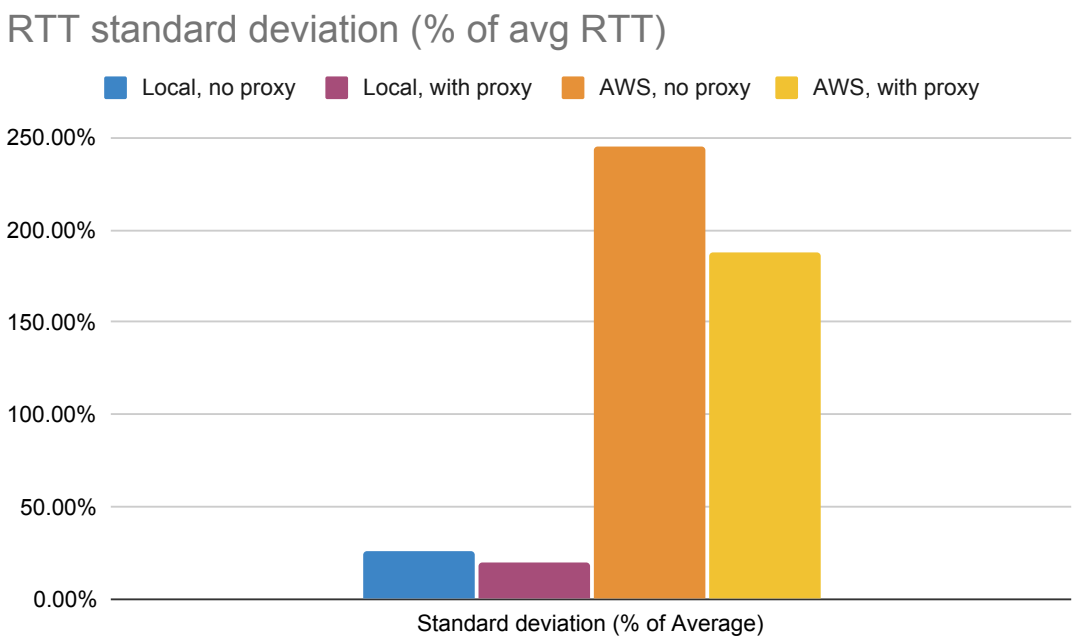

Figure 5.7: The standard deviation as a percentage of the average latency. Lower values are better.

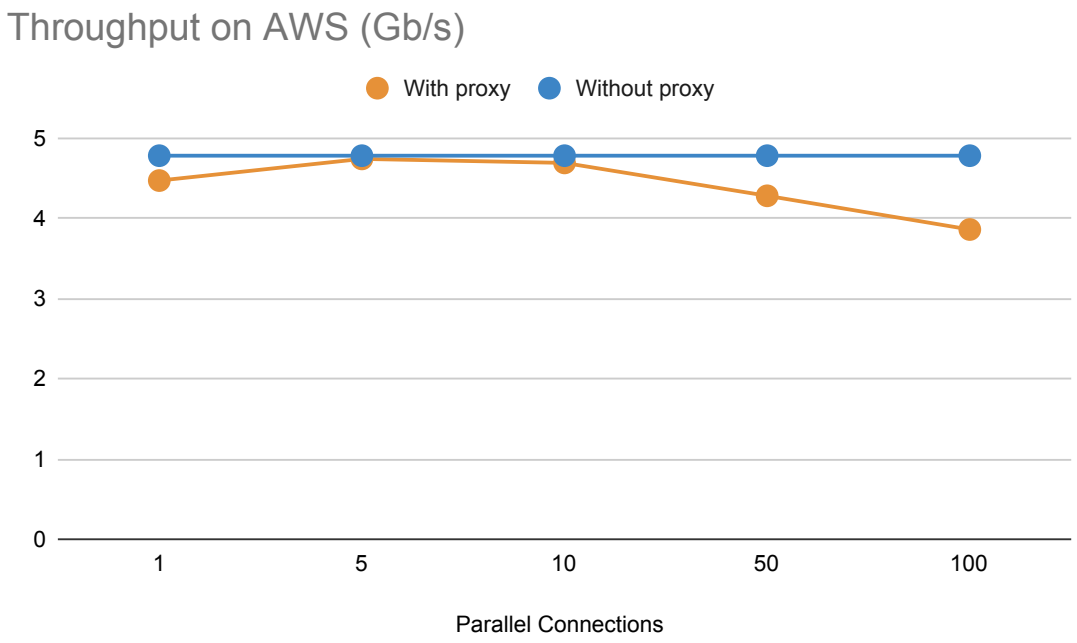

Figure 5.8: The measured throughput between two AWS VMs. A smaller difference between "with proxy" and "without proxy" is better.

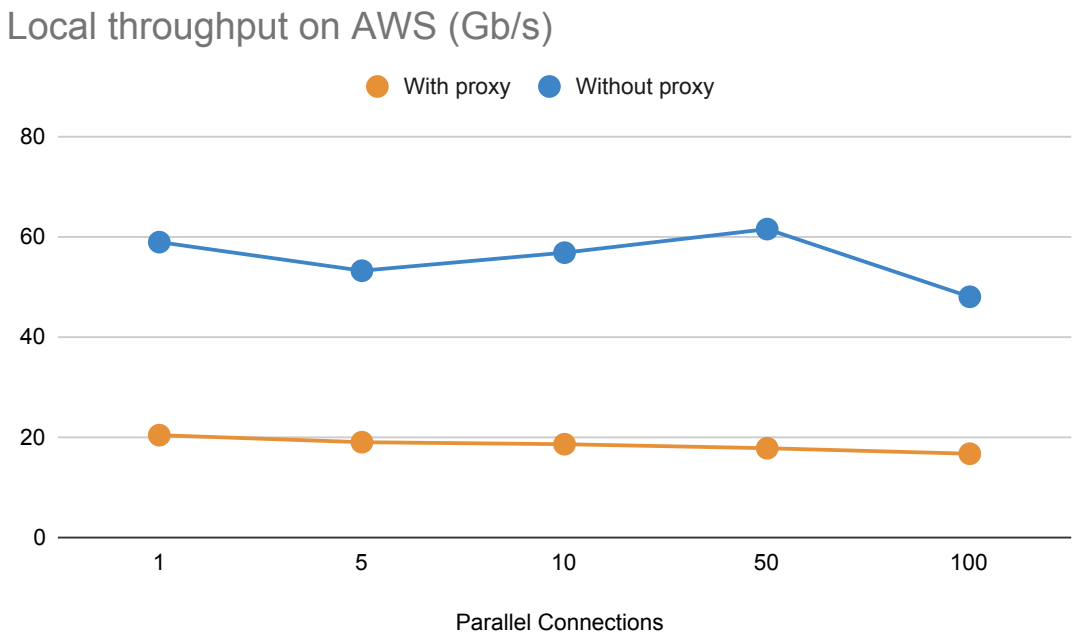

Figure 5.9: The measured local throughput of an AWS VM. A smaller difference between "with proxy" and "without proxy" is better.

6

Discussion & Conclusion

This thesis aimed to design a new moving-target defense strategy that mitigates cross-VM side-channel attacks. The aim of the design work was to provide a solution that is easy to integrate into existing systems. The idea has successfully been implemented in software, demonstrating that this type of moving target defense is a feasible option for extending the security precautions of a sensitive cloud application.

6.1 Discussion

The tests involving Node Exporter, Prometheus, and Grafana show the usability of this system, while at the same time highlighting the major issue with moving-target defense that involves migrations between different physical machines.

As seen in Figure 5.1, the service is not reachable 100% of the time, even with migration strategy B, which is aimed at uptime. The cause is that the handover between the old VM and the new VM happens once the new VM is ready. At the moment that the new VM becomes ready, the application inside the VM still needs to start. As a consequence, the proxy cannot forward any data from the clients to the server after the handover.

The migration time on AWS EC2 is relatively long, between one and two minutes. How much impact this has depends on the application, but for interactive applications, the downtime or loss of state is a significant issue and should be reduced if possible. For a stateless application like a web server, this is a non-issue.

While the moving-target defense is relatively easy to integrate into a cloud architecture, considerable thought needs to be put into if the service is suitable for MTD, and with which migration strategy.

The throughput over the network with and without the proxy is very similar. The obvious bottleneck is the 5Gbps network card of the hardware. At around 20Gbps, as seen in the unbounded local test in Figure 5.9, the bottleneck moves. It is unclear if the bottleneck moves to the CPU or some internal bus. However, 20Gbps full-duplex is a lot of data for a single VM, and any hardware with such a capable network card is usually paired with a lot more capable CPU, meaning this is likely a non-issue.

The latency, both on AWS and local, is almost double when there is a proxy present,

as shown in Figure 5.4. The reason for the extra latency is the additional traversal of the network stack when there is a proxy in between. The amount of traversals of the network stack doubles with a proxy, and since there is no application latency in these benchmarks, the numbers are dominated by the latency of the network stack. The additional latency in absolute numbers, as shown in Figure 5.5 is less than one millisecond. Comparing that to a typical query of a PostgreSQL database which takes from 1 millisecond to hundreds of milliseconds, it is clear that the added latency of the proxy is insignificant for most applications.

6.2 Future work

A shortcoming of the current implementation is that the manager will initiate a handover when the new virtual machine has properly started. The manager, however, does not take into account that the application itself also needs time to start, resulting in unexpected downtime. This can easily be fixed by tweaking the handover logic, either by delaying the handover for a predetermined amount of seconds that gives the applications time to start up, or by implementing a check to make sure the application is ready. The latter would likely require application specific modifications though, which is to be avoided if this solution is meant to be a drop-in solution.

For this thesis, the moving-target defense has only been implemented on AWS. Other cloud platforms have been left out for simplicity. However, not everybody uses AWS, as there are still plenty of applications that run on Azure, GCP or in a self-managed environment. To increase the usability of the manager, support for the other major cloud platforms and for commercial hypervisors for self-managed environments like VMware could be added.

Many cloud setups already have a static component to manage multiple deployments of a service for redundancy and scale in the form of an orchestration tool like Kubernetes. To avoid complicating the setup, the manager could be integrated into such orchestration tools, as these integration tools already have the ability to start additional instances of a service on common hypervisors or cloud platforms. An additional benefit of integrating the manager with orchestration tools is that it avoids having to implement support for cloud platforms or hypervisors, as that is something that the orchestration tool is already capable of managing.

To solve the issue of long migration times, a cloud application could consider running the software inside a container or microVM, as that can result in a significant speedup of the migration time. Platforms that offer faster migration time are docker containers that offer migration times of less than five seconds [10] or Firecracker micro virtual machines [25], a VM platform designed to be as fast as containers. Reducing the migration time from minutes to seconds significantly reduces the downtime or state loss for any application using moving-target defense, but to support those platforms, additional work on the manager is required.

6.3 Conclusion

It has been shown that employing moving-target defense to mitigate cross-VM side-channel attacks is a viable strategy. A prototype was developed to showcase the usability and benchmark the performance. The results show that making moving-target defense part of the cloud deployment is a viable option. However, there are certain drawbacks. It needs to be considered which of the two proposed migration strategies, aimed at consistency or availability, is most appropriate for the situation. In addition, implementing moving-target defense as proposed in this thesis does increase the complexity slightly. It needs to be determined per application whether the trade-off between the increased costs and network overhead versus the additional security is worth it. For applications where requirements of security are above those of performance and costs, adding moving-target defense as proposed in this thesis is most likely worth it.

Bibliography

[1] A. Aljabre, "Cloud computing for increased business value," *International Journal of Business and social science*, vol. 3, no. 1, 2012.

[2] Y. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart, "Cross-vm side channels and their use to extract private keys," 2012. DOI: 10.1145/2382196.2382230.

[3] J.-H. Cho, D. P. Sharma, H. Alavizadeh, *et al.*, "Toward proactive, adaptive defense: A survey on moving target defense," *IEEE Communications Surveys & Tutorials*, vol. 22, no. 1, pp. 709–745, 2020. DOI: 10.1109/COMST.2019.2963791.

[4] R. Zhuang, S. A. DeLoach, and X. Ou, "Towards a theory of moving target defense," in *Proceedings of the First ACM Workshop on Moving Target Defense*, ser. MTD '14, Scottsdale, Arizona, USA: Association for Computing Machinery, 2014, pp. 31–40, ISBN: 9781450331500. DOI: 10.1145/2663474.2663479. [Online]. Available: https://doi.org/10.1145/2663474.2663479.

[5] G. J. Popek and R. P. Goldberg, "Formal requirements for virtualizable third generation architectures," *Commun. ACM*, vol. 17, no. 7, pp. 412–421, Jul. 1974, ISSN: 0001-0782. DOI: 10.1145/361011.361073. [Online]. Available: https://doi.org/10.1145/361011.361073.

[6] A. Greenberg, *What is a side channel attack?* Jun. 2020. [Online]. Available: https://www.wired.com/story/what-is-side-channel-attack/.

[7] V. Varadarajan, T. Ristenpart, and M. Swift, "Scheduler-based defenses against cross-vm side-channels," 2014.

[8] O. Aciicmez and J.-P. Seifert, "Cheap hardware parallelism implies cheap security," pp. 80–91, 2007. DOI: 10.1109/FDTC.2007.16.

[9] J. Xiao, Z. Xu, H. Huang, and H. Wang, "Security implications of memory deduplication in a virtualized environment," in *2013 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2013, pp. 1–12. DOI: 10.1109/DSN.2013.6575349.

[10] M. Buck and P. Tibom, M.S. thesis, 2022. [Online]. Available: https://odr.chalmers.se/items/19535ce6-4924-49b9-ab10-76df469f15c5.

[11] A. Ephremides, J. Wieselthier, and D. Baker, "A design concept for reliable mobile radio networks with frequency hopping signaling," *Proceedings of the IEEE*, vol. 75, no. 1, pp. 56–73, 1987. DOI: 10.1109/PROC.1987.13705.

[12] N. Golmie, O. Rebala, and N. Chevrollier, "Bluetooth adaptive frequency hopping and scheduling," in *IEEE Military Communications Conference, 2003. MILCOM 2003.*, vol. 2, 2003, 1138–1142 Vol.2. DOI: 10.1109/MILCOM.2003.1290352.

[13] N. O. Ahmed and B. Bhargava, "Mayflies: A moving target defense framework for distributed systems," in *Proceedings of the 2016 ACM Workshop on Moving Target Defense*, ser. MTD '16, Vienna, Austria: Association for Computing Machinery, 2016, pp. 59–64, ISBN: 9781450345705. DOI: 10.1145/2995272.2995283. [Online]. Available: https://doi.org/10.1145/2995272.2995283.

[14] E. Al-Shaer, Q. Duan, and J. H. Jafarian, "Random host mutation for moving target defense," in *Security and Privacy in Communication Networks*, A. D. Keromytis and R. Di Pietro, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 310–327, ISBN: 978-3-642-36883-7.

[15] T. E. Carroll, M. Crouse, E. W. Fulp, and K. S. Berenhaut, "Analysis of network address shuffling as a moving target defense," in *2014 IEEE International Conference on Communications (ICC)*, 2014, pp. 701–706. DOI: 10.1109/ICC.2014.6883401.

[16] K. A. Torkura, M. I. Sukmana, A. V. Kayem, F. Cheng, and C. Meinel, "A cyber risk based moving target defense mechanism for microservice architectures," in *2018 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Ubiquitous Computing & Communications, Big Data & Cloud Computing, Social Computing & Networking, Sustainable Computing & Communications (ISPA/IUCC/BDCloud/SocialCom/SustainCom)*, 2018, pp. 932–939. DOI: 10.1109/BDCloud.2018.00137.

[17] *Huge cloud market still growing at 34% per year; amazon, microsoft & google now account for 65% of the total | synergy research group*, Apr. 2022. [Online]. Available: https://www.srgresearch.com/articles/huge-cloud-market-is-still-growing-at-34-per-year-amazon-microsoft-and-google-now-account-for-65-of-all-cloud-revenues.

[18] [Online]. Available: https://aws.amazon.com/developer/tools/.

[19] "Rfc: The tls protocol version 1.0." (), [Online]. Available: https://datatracker.ietf.org/doc/html/rfc2246.

[20] R. Pereira, M. Couto, F. Ribeiro, *et al.*, "Ranking programming languages by energy efficiency," *Science of Computer Programming*, vol. 205, p. 102 609, 2021, ISSN: 0167-6423. DOI: https://doi.org/10.1016/j.scico.2021.102609. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167642321000022.

[21] N. S. Agency, *Cybersecurity information sheet*, 2022. [Online]. Available: https://media.defense.gov/2022/Nov/10/2003112742/-1/-1/0/CSI_SOFTWARE_MEMORY_SAFETY.PDF.

[22] D. J. Adrian Taylor Andrew Whalley and C. s. t. Nasko Oskov, *An update on memory safety in chrome*, 2021. [Online]. Available: https://security.googleblog.com/2021/09/an-update-on-memory-safety-in-chrome.html.

[23] "Tokio - an asynchornous rust runtime." (), [Online]. Available: https://tokio.rs/.

[24] "Iperf - the tcp, udp and sctp network bandwidth measurement tool." (), [Online]. Available: https://iperf.fr.

[25] "Firecracker microvm." (), [Online]. Available: https://firecracker-microvm.github.io/.

A

Appendix 1

The source code for the implementations of the MTD manager and the proxy are hosted on GitHub. Links to the respective repositories can be found below, referred to by their respective development code-names.

Polemos, the MTD manager
`https://github.com/thefeli73/polemos`

Proxima-Centauri, the run-time configurable reverse TCP proxy
`https://github.com/GreenPenguino/proxima-centauri`