PURESEC

New Attack Vector: Serverless Crypto-Mining

Hackers can now turn a single vulnerable serverless function into hundreds to thousands of compute instances running virtual crypto-mining farm, a PureSec research shows. For the attacker, serverless is becoming an extremely cost effective "playground", enabling auto-scaling a successful attack. For the victim, financial damage could reach as much as $120,000 per month. The scalable nature of serverless architectures, which is precisely what's driving the rapid adoption of the new technology, has a dark side. In this article, we will introduce the newly discovered threat and attack vector, and offer solutions to defend your serverless application.

Traditional Cloud Crypto-mining Attacks

In recent months, several articles and blog posts exposed how malicious attackers are abusing cloud environments in order to infect them with crypto-mining malware. As an example, in February 2018, cybersecurity firm Redlock reported that hackers had secretly infiltrated public cloud environments and were using the compute instances to mine cryptocurrencies.
Among the companies that were hit by this attack was automotive manufacturer Tesla.

Up until recently, in order to run cost-effective crypto-mining operations in the cloud, attackers had to infect a large array of servers or cloud-based server instances. As an example, attackers would scan millions of web servers looking for remote code execution (RCE) vulnerabilities, which would allow them to install crypto-mining malware remotely. Locating large arrays of vulnerable servers that could run crypto-mining malware is a daunting task, which may attract attention and risk its operators. Several such web scanning and exploitation campaigns were spotted and reported recently by Imperva and Sucuri (pdf).

Crypto-mining In The Age Of Serverless: A New Attack Vector

One of the key benefits of serverless architectures is their ability to continuously scale the application by running copies of the same serverless function code in response to each additional event trigger. Serverless code runs in parallel and processes each event trigger individually, scaling precisely with the size of the workload (see the “References (I)” section for additional information regarding auto-scaling in major serverless platforms).

But doesn’t this auto-scaling mean that “serverless-assisted” attacks are just as scalable?

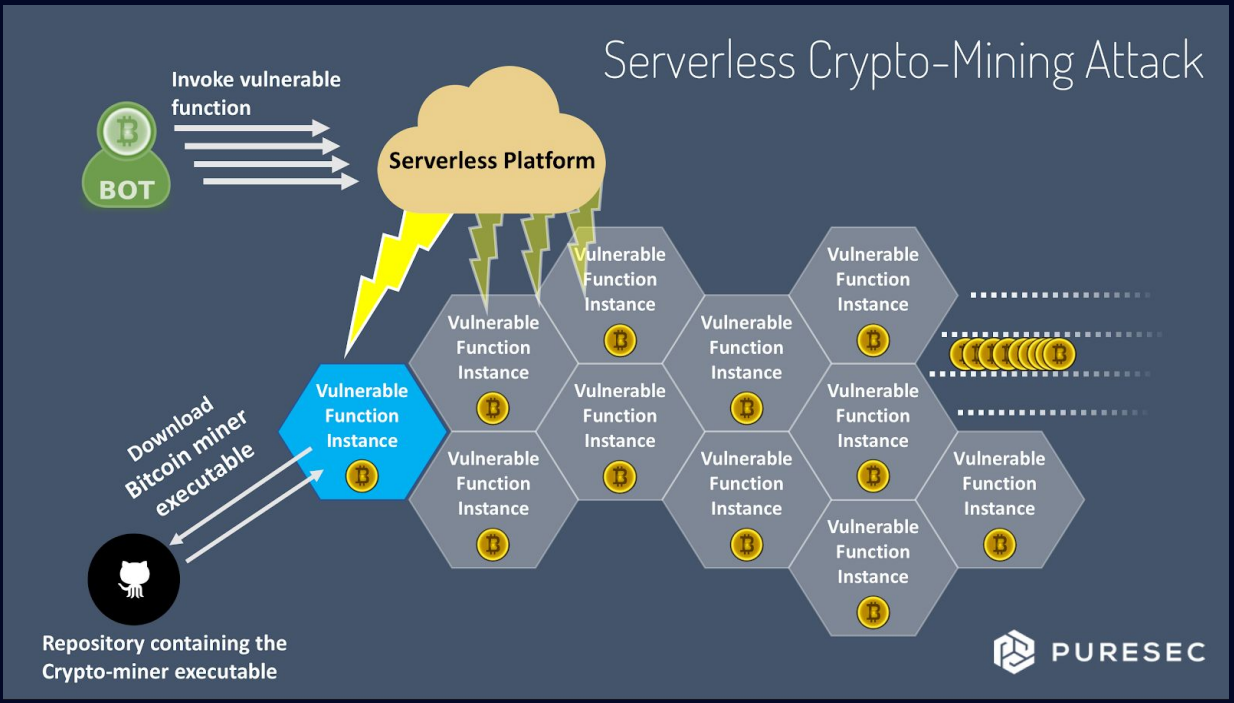
PureSec’s research team examined this question with three leading public-cloud serverless platforms. The team succeeded in forcing functions, which were vulnerable to remote code execution, to download an off-the-shelf crypto-miner executable during function execution, and to perform crypto-mining computations, in parallel to the function’s normal execution tasks.

During the simulated attack, the team affected the execution process in a way that caused the serverless platforms to scale until they reached their concurrent execution limit. To put it in layman terms, the team managed to abuse a single vulnerable serverless function and by harnessing the auto-scaling nature of serverless platforms, to turn the single function into a “virtual crypto-mining farm” capable of producing meaningful cryptomining results that are cost effective for the attacker.

The ability to exploit and harness serverless platforms to execute crypto-mining malware provides several key benefits for malicious attackers:

1. One vulnerability is enough. Attackers only need to find a single vulnerable serverless function and abuse the inherent auto-scaling nature of the serverless platform in order to mimic the work of hundreds to thousands of machines working in parallel. Moreover, since attackers only need a single vulnerable function, the task of finding a suitable target requires less resources, and attracts less attention.
2. Impossible to protect against, using traditional security solutions. Many serverless consumers are still struggling with application security of their serverless functions. This allows attackers to perform crypto-mining activities under the radar, without being spotted. Traditional application layer protections, such as those that are deployed on servers and networks, are irrelevant for serverless architectures. (This goes for Web Application Firewalls, Runtime Application Self-Protection, Intrusion Detection and other malware prevention solutions as well). As long a the serverless application is not secure, attackers are free to wreck havoc and abuse functions without running into any issues or getting detected.

With commonly used serverless platform account configurations, and in relation to concurrent execution limits, the financial loss for the owner of the vulnerable serverless function, as a result of a single attack, might reach as much as $120,000 USD per month



How To Mitigate The Risk?

The only way to protect your application from Serverless Crypto-Mining Attacks is using an SSRE (Serverless Security Runtime Environment).

Just like any other type of software, your serverless functions may be vulnerable to application layer attacks. If your functions contain vulnerabilities, malicious users and hackers will quickly find and exploit them in order to tamper or steal sensitive data, damage your application, deny service from other users and so forth. In order to withstand such attacks, your serverless functions require a solution for protecting them against application layer attacks.

PureSec recently launched its Serverless Security Runtime Environment (SSRE) Beta program for AWS Lambda and Azure Functions.

Enterprises can protect their serverless functions by deploying an SSRE solution that automatically blocks any attempt to inject and execute crypto-mining malware targeting serverless applications.

If you want to learn more, visit our web site at: https://www.puresec.io

References

I. **The following table summarizes the auto-scaling behavior and default limits, as defined** in the official documentation of AWS Lambda, IBM OpenWhisk, Google Cloud Functions and Microsoft Azure Functions:

SERVERLESS PLATFORM	AUTO-SCALING BEHAVIOR	DEFAULT MAX. CONCURRENT EXECUTIONS LIMIT
AWS Lambda	AWS Lambda will dynamically scale capacity in response to increased traffic, subject to your account's Account Level Concurrent Execution Limit. To handle any burst in traffic, Lambda will immediately increase your concurrently executing functions by a predetermined amount, dependent on which region it's executed (e.g. US-EAST-1: 3,000)	By default, the concurrent execution limit is enforced against the sum of the concurrent executions of all functions. The shared concurrent execution pool is referred to as the unreserved concurrency allocation. The default is set to 1,000.
IBM Cloud Functions	Hard limit for each “namespace”	The number of activations that are either executing or queued for execution for a Namespace cannot exceed 1000. This limit value is fixed, but can be increased if a business case can justify higher safety limit values.
Google Cloud Functions	Cloud Functions invoked by HTTP scale quickly to the desired invocation rate, while background functions (non-HTTP triggered) scale more gradually. In the latter case, scalability depends on the duration of functions, and longer functions will scale slightly slower.	1,000,000 function invocations per 100 seconds (any kind of function) Max 1000/second concurrent invocations for a background function.

SERVERLESS PLATFORM	AUTO-SCALING BEHAVIOR	DEFAULT MAX. CONCURRENT EXECUTIONS LIMIT
 **Microsoft Azure Functions**	When multiple triggering events occur faster than a single-threaded function runtime can process them, the runtime may invoke the function multiple times in parallel. If a function app is using the Consumption hosting plan, the function app could scale out automatically. Each instance of the function app, whether the app runs on the Consumption hosting plan or a regular App Service hosting plan, might process concurrent function invocations in parallel using multiple threads. The maximum number of concurrent function invocations in each function app instance varies based on the type of trigger being used as well as the resources used by other functions within the function app.	Unspecified explicitly, seems to depend on different factors and to vary based on event types.