Vol.2017-CSEC-77 No.13
Vol.2017-IOT-37 No.13
2017/5/26

彼はクラウドを愛したが、クラウドは彼を愛さなかった

柏崎 礼生[1,a)]

概要：クラウドコンピューティングの登場により人々は強大な計算機資源を手に入れることができるようになった。クラウドコンピューティングをクラウドコンピューティングたらしめる属性の一つに従量課金がある。人々が強大な計算機資源を手に入れることと、その代償となる資本は等価交換される。一方で人々は時として意図せず強大な資源を利用し、意図しない代償の請求に苦悩することがある。本稿では意図しない利用によりクラウドコンピューティングプロバイダから 1 ヶ月で約 580 万円を請求された哀れな一個人の事例を紹介するとともに、その原因を説明し、意図と資本の齟齬を解消し得るモデルを提案する。

He loved the cloud, but the cloud did not love him.

Hiroki Kashiwazaki[1,a)]

Abstract: People can obtain a huge and a strong power of computing resources by a raise of cloud computing environments. Pay as you go is one of the identical definition of cloud computing environments. If a person uses the large amount of cloud computing resources, then he must be charged. Meanwhile a person sometimes can meet the great amount of bill and then be embarrassed. This paper shows a pitiful person who were charged more than 50 thousands dollars as one month usage fee by a certain cloud computing provider. And then also the paper explains the reason of the incident, and proposes the model that can be solve this sort of contradiction between users and providers.

Keywords: cloud computing environment, pay as you go, security, API, design

1. Background

According NIST Special Publication 800-145 [1], "measured service" is one of essential characteristics of cloud computing. In the publication, measured service is explained as follows: "Cloud systems automatically control and optimize resource use by leveraging a metering capability at some level of abstraction appropriate to the type of service (e.g., storage, processing, bandwidth, and active user accounts)". And as the footnote of the "capability" explain it as "Typically this is done on a pay-per-use or charge-per-use basis". "Pay-per-use" also known as "Pay as you go" is one of the identical definition of cloud computing. "If you use it, a bill will come[*1]." But sometimes users can meet an unexpected amount of billing statement. Reasons of absence of expectation can be various. If the billing is unintended, the contradiction of intention between users and providers make both of them unhappy.

Fortunately (or unfortunately), the author of this paper met an unexpected amount of billing statement from Amazon Web Service (AWS) on 3rd, February, 2017. When he got a mail of the statement at that time, he could not understand the numerical value described in the mail. He usually utilized only Amazon Elastic Block Store (EBS) Service and an usual monthly amount of billing statement was less than 1 U.S. dollar. At that time,

[1] 大阪大学
Osaka University
[a)] reo@cmc.osaka-u.ac.jp

[*1] Phil Alden Robinson: Field of Dreams (1989)

in that mail, there was an unrealistic numerical number "JPY 5,797,028". It was equal to USD 51,125 and also approximately equal to his annual earnings. At first, he considered the statement was some kinds of mistakes of AWS beacause he has not used AWS, especially charged services, for a long time. Soon after the consideration, he noticed that he wrote some codes using Amazon Product Advertising API during a new year holidays. But he also reminded that he enabled his AWS Multi-Factor Authentication (MFA) and AWS Identity and Access Management (IAM) at that time. So he foolishly turned to feel easy because he could not find any other vulnerabilities of his AWS account at all and he had proudly confidence in no fault of him.

Can clever readers guess or infer causation of the billing? In the next section, this paper show causation in chronological order.

2. Causation

The causation date back to 2009. The author have written a tiny code using Amazon E-Commerce Service (Amazon ECS) to manage his tons of books and motion picture contents. Though the origin of Amazon ECS can not be found clearly, according to GitHub repository, a first commit of amazon-ecs rubygems package[*2] have committed in June 2009. Also according to RubyGems, a first version of amazon-ecs was 0.5.0, published in December, 2006. A book written by Jason Levitt was published in 2005, whose title includes "Amazon E-Commerce Service" [2]. It is a probable thing that Amazon ECS launched before 2005.

2.1 amazon-ecs

According to amazon-ecs 0.5.0, users can only set their `:aWS_access_key_id` as `Amazon::Ecs.options` and query words. Then users can get the result from Amazon.com. The identification of `access_key_id` was unique to a user and was used to the associate program of Amazon.com. If users publish the ID (and other users can use the ID), the users can earn more rewards from the associate program. In 2009, Amazon.com changed their services and name of the services. Amazon ECS, also known as Amazon Associate Web Service was altered to "Product Advertising API". Product Advertise API turned to need a digital signature of users to authenticate per a request since 15 August, 2009. The package of amazon-ecs was also updated in keeping with the change since version 0.5.5 on 17 July, 2009 (図 1, 2).

```
Amazon::Ecs.options =
  {:aWS_access_key_id => [your developer token]}
res =
  Amazon::Ecs.item_search(
    'ruby',
    {:response_group => 'Medium',
     :sort => 'salesrank'})
```

図 1 an example code of amazon-ecs 0.5.0

```
Amazon::Ecs.configure do |options|
  options[:aWS_access_key_id] = [your access key]
  options[:aWS_secret_key] = [you secret key]
end
res =
  Amazon::Ecs.item_search(
    'ruby',
    {:response_group => 'Medium',
     :sort => 'salesrank'})
```

図 2 an example code of amazon-ecs 0.5.5

Accidentally, the author lost his interest to manage his library and felt a loss of motivation to maintain his tiny code. So he stopped to continue to update his code since 18 February, 2009, according a log of git. The next time when he turned to maintain his old code was on 1 January, 2017. During his new year holidays in 2017, he started to renew his code. He found that there were a lot of changes around Amazon Product Advertising API. First of all, Multi-Factor Authentication (MFA) was introduced to login to a console of AWS.

2.2 Multi-Factor Authentication

Generally, a multi-factor authentication is a method of computer access control in which a user is granted access only after successfully presenting several separate pieces of evidence to an authentication mechanism. And typically at least two of the following categories is needed. Knowledge (something they know), possession (something they have), and inherence (something they are). A purpose of MFA was to provide a simple best practice that can add an extra layer of protection on top of users' credentials. With MFA enabled, when a user signs in to an AWS website, they will be prompted for their user name and password (the first factor—what they know), as well as for

[*2] `https://github.com/jugend/amazon-ecs`

	Virtual MFA Device	Hardware Key Fob MFA Device	Hardware Display Card MFA Device	SMS MFA Device
Device	See table below.	Purchase.	Purchase.	Use your mobile device.
Physical Form Factor	Use your existing smartphone or tablet running any application that supports the open TOTP standard.	Tamper-evident hardware key fob device provided by Gemalto, a third-party provider.	Tamper-evident hardware display card device provided by Gemalto, a third-party provider.	Any mobile device that can receive Short Message Service (SMS) messages.
Price	Free	$12.99	$19.99	SMS or data charges may apply.
Features	Support for multiple tokens on a single device.	The same type of device used by many financial services and enterprise IT organizations.	Similar to key fob devices, but in a convenient form factor that fits in your wallet like a credit card.	Familiar option with low setup costs.
Compatibility with Root Account	✔	✔	✔	
Compatibility with IAM User	✔	✔	✔	✔

図 3 A diagram of Amazon Multi-Factor Authentication (MFA) form factors

an authentication code from their AWS MFA device (the second factor—what they have). Taken together, these multiple factors provide increased security for your AWS account settings and resources. Users can enable MFA for their AWS account and for individual IAM users that they have created under their account. MFA can be also be used to control access to AWS service APIs[*3].

According to a first post concerning to MFA to AWS Security Blog, MFA was launched before at least 30 April, 2013. Four kinds of form factors of MFA is provided (Virtual MFA Device, Hardware Key Fob MFA Device, Hardware Display Card MFA Device, and SMS MFA Device). Virtual MFA Device is the only way to use MFA completely freely (SMS MFA Device may cost SMS or data charges). Virtual MFA Applications are provided on several platforms including Android, iPhone, Windows Phone and Blackberry (Fig.3).

2.3 Identity and Access Management

AWS Identity and Access Management (IAM) was also introduced at the nearly same time. According to a first post concerning to IAM to the security blog, IAM was launched before at least 6 May 2013. And also according to Japanese Amazon Web Service Blog, IAM was launched on 3 May, 2011. IAM is an implementation to separate privileges and enable user to securely control access to AWS services and resources. Using IAM, users can create and manage AWS users and groups, and use permissions to allow and deny their access to AWS resources[*4]. With IAM, users can create multiple IAM users under the umbrella of their AWS account or enable temporary access through identity federation with their corporate directory. In some cases, users can also enable access to resources across AWS accounts. Without IAM, users must either create multiple AWS accounts, each with its own billing and subscriptions to AWS products, or their employees must share the security credentials of a single AWS account. In addition, without IAM, users cannot control the tasks a particular user or system can do and what AWS resources they might use[*5].

The following enumeration describes the canonical use case for creating an IAM user[*6].

(1) create user
(2) give user security credentials
(3) put user into one or more groups
(4) give user a login profile (optional)

To enable IAM to use Amazon Product Advertise API, users must make a specific user for its use, then allow access with using API, CLI and developers tools including SDK to enable both of `access_key_id` and `secret_access_key`. Then set privileges to the user. At the time of 1 Jan. 2017, there were no documents concerning to suitable setting of policies to use Product Advertise API. So users had no candidates without using the role “AdministratorAccess” or “PowerUserAccess” at that time. After setting policies to the IAM user, users can get the credential information of `access_key_id` and `secret_access_key`.

2.4 GitHub

GitHub[*7] is a web-based Git or version control repository and Internet hosting service. It offers all of the distributed version control and source code management functionality of Git as well as adding its own features.

*3 https://aws.amazon.com/iam/details/mfa/
*4 https://aws.amazon.com/iam/
*5 http://docs.aws.amazon.com/IAM/latest/UserGuide/getting-setup.html
*6 https://aws.amazon.com/iam/details/manage-users/
*7 https://github.com/

It provides access control and several collaboration features such as bug tracking, feature requests, task management, and wikis for every project[*8]. Because the author have managed the tiny code using Amazon Product Advertise API, he created new repository on his account on GitHub, then he push the altered code to GitHub on Jan 1 13:35:09 (GMT). The code included all the information of `:AWS_access_key_id`, `:AWS_secret_key`, and `:associate_tag`. But he found that the code was not secure. Two days ago, he altered his code. Then he committed and pushed to GitHub on Jan 3 7:32:16 2017 (GMT). This version of code concealed the credentials to an external local file. The main part of the code shows below.

```
OPTS = Hash.new
OPTS[:configfile] = "conf.yaml"
CONF = YAML.load_file(OPTS[:configfile])
Amazon::Ecs.configure do |options|
  options[:aWS_access_key_id]
    = CONF[:AWS_accesss_key_id]
  options[:aWS_secret_key] = CONF[:AWS_secret_key]
  options[:associate_tag] = CONF[:associate_tag]
  options[:country] = CONF[:country]
end
```

図 4 a part of code with the latest amazon-ecs

One month later since the push to GitHub, the author got the billing statement of January 2017 from AWS. Then and only then, he was sure to be fault of AWS and have never find his fault because he enforce strength of his account by enabling MFA. He could not find his fault of IAM at that time.

3. Response

Firstly, the author called to the customer support of AWS and he claim his justice and injustice of the billing. A staff of the support searched Elastic Compute Cloud (EC2) usage report of the author and the staff made sure of authentic usage history of EC2 instances whether accidental or intentional. The staff told the author to make sure the usage report and to stop the active instances. The author followed the instruction and made sure the usage and existence of EC2 instances. A number of the instance was 12 and all of the instance was c3.8xlarge and c4.8xlarge. c3.8xlarge instance consists of 32 vCPUs, 60 GiB memory and two 320 GB SSD Storage. c4.8xlarge instance consists of 36 vCPUs, 60 GiB memory and 4 Gbps dedicated Elastic Block Store (EBS) Bandwidth. C3 and C4 instance are compute optimized instant types. C4 instances are the latest generation of Compute-optimized instances, featuring the highest performing processors and the lowest price/compute performance in EC2[*9]. All of instances were locked and not deleted collectively. The author deleted all of the instance manually.

3.1 EC2 usage report

Even at that time, the author could not find reasons of illegal use. He was engaged at that time, he reported the incident to his fiancée. Then she showed him an URI concerning to illegal use of AWS and billing 6,000 USD[*10]. Firstly, he laughed off the information, but immediately after the laughing, he found that the information was quite correspond to his incident. He apologized her and told the staff the reason of illegal accesses. On the repository of GitHub, the code including secret key was still there in the "history" of the repository. He deleted the repository and also deleted the IAM user correspond to the key.

According to the EC2 usage report of the author, between 30 and 90 minutes after his pushing to GitHub, secret key was used by malicious users. As previously mentioned, the commitment was done on Jan 1 13:35:09 (GMT). Figure 5 shows usage log of Amazon EC2 around the illegal usage. Until 1 Jan. 2017 14:00:00 (GMT), the log only shows "CreateVolume" operation that means only (legal uses of) EBS were there. After 14:00:00, 3 SportUsages of c4.8xlarge, 3 SpotUsages of c3.8xlarge and 1 BoxUsage of c4.8xlarge was started for the first 1 hour. SpotUsages of c4.8xlarge and c3.8xlarge was finished by 00:00:00 2 Jan. 2017 (GMT).

3.2 billing statement

Figure 6 shows the billing statement of AWS in Jan. 2017. A dominant share of EC2 usage (92.6%) can be found as a characteristics of the billing. Billing of data transfer can be negligible (0.007%). Some research colleague supposed that the illegal usage of EC2 may be worked to Bitcoin mining. If the author would not delete and would save the instances just after instruction from the staff of AWS, the author could search the instances in detail (but he could not).

[*8] https://en.wikipedia.org/wiki/GitHub

[*9] https://aws.amazon.com/ec2/instance-types/

[*10] http://qiita.com/mochizukikotaro/items/a0e98ff0063a77e7b694

Service	Operation	UsageType	Resource	StartTime	EndTime	UsageValue
AmazonEC2	CreateVolume	USW2-EBS:VolumeUsage		01/01/17 08:00:00	01/01/17 09:00:00	8589934592
AmazonEC2	CreateVolume	USW2-EBS:VolumeUsage		01/01/17 09:00:00	01/01/17 10:00:00	8589934592
AmazonEC2	CreateVolume	USW2-EBS:VolumeUsage		01/01/17 10:00:00	01/01/17 11:00:00	8589934592
AmazonEC2	CreateVolume	USW2-EBS:VolumeUsage		01/01/17 11:00:00	01/01/17 12:00:00	8589934592
AmazonEC2	CreateVolume	USW2-EBS:VolumeUsage		01/01/17 12:00:00	01/01/17 13:00:00	8589934592
AmazonEC2	CreateVolume	USW2-EBS:VolumeUsage		01/01/17 13:00:00	01/01/17 14:00:00	8589934592
AmazonEC2	EBS:Gp2-IO-Write	USE2-EBS:VolumeIOUsage.gp2		01/01/17 14:00:00	01/01/17 15:00:00	439117
AmazonEC2	RunInstances	USW2-C3DataTransfer-In-Bytes		01/01/17 14:00:00	01/01/17 15:00:00	38685
AmazonEC2	EBS:Gp2-IO-Write	EU-EBS:VolumeIOUsage.gp2		01/01/17 14:00:00	01/01/17 15:00:00	24003
AmazonEC2	EBS:Gp2-IO-Read	USE2-EBS:VolumeIOUsage.gp2		01/01/17 14:00:00	01/01/17 15:00:00	323844
AmazonEC2	RunInstances:0002:SV051	USE2-SpotUsage:c4.8xlarge		01/01/17 14:00:00	01/01/17 15:00:00	12
AmazonEC2	EBS:Gp2-IO-Write	EBS:VolumeIOUsage.gp2		01/01/17 14:00:00	01/01/17 15:00:00	317164

図 5 usage log of Amazon Elastic Comput Cloud (EC2)

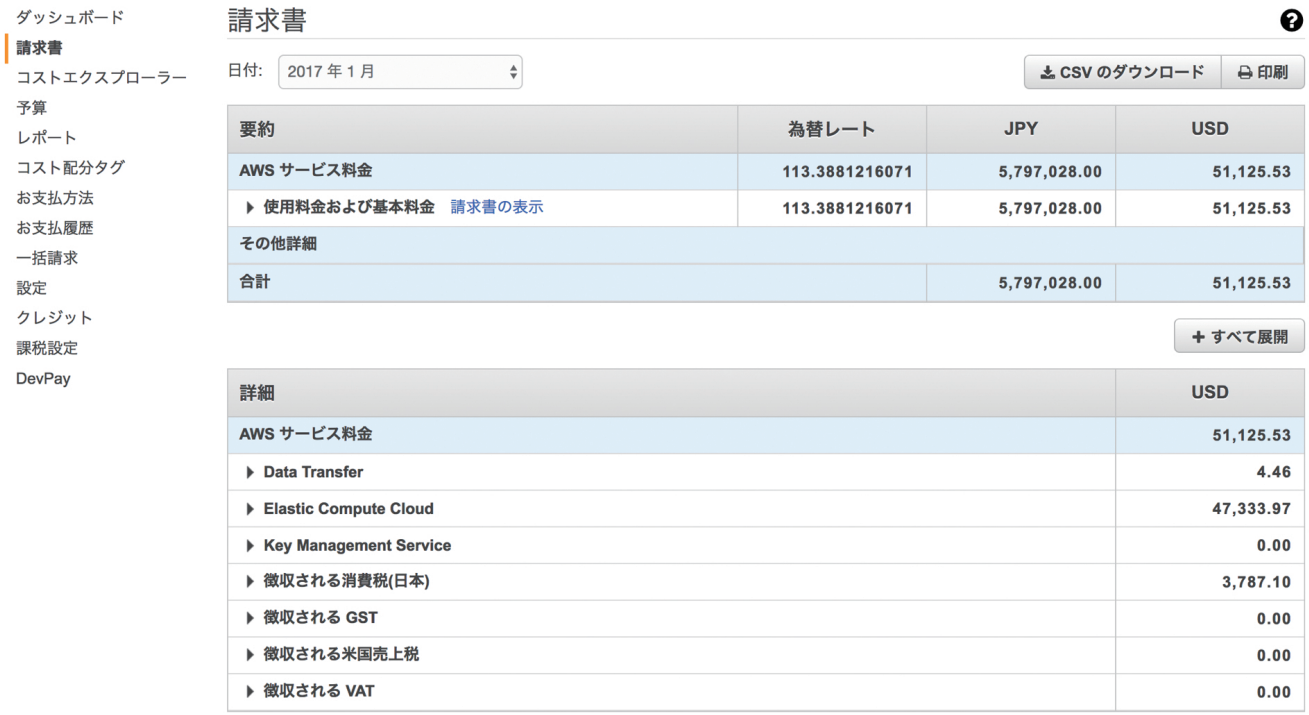

図 6 a billing statement of Amazon Web Service in January 2017

4. Reflection

The main causation of the incident and the worst fault of the author was to upload credentials to public space (GitHub). That is all. Meanwhile, a lack of documents can be pointed out. AWS have never provide sufficient documents concerning to suitable setting of IAM policies to Productive Advertise API. So some user found that "AdministratorAccess" or "PowerUserAccess" policies were only way to use Productive Advertise API and they wrote the information to their public document such as technical blogs. Though the information was propagated around world wide web, AWS have never updated the documents concerning to Product Advertising API. After the incident, AWS updated the document "Becoming a Product Advertising API Developer"[*11] on 10 Feb. 2017. According to the document, users must write the policy document manually, not by selecting a policy type. Until a publication of the document, no users on the earth has any way to know such kind of detail description of policy document.

AWS also provide "CloudTrail" service to log application programming interfaces (APIs) call. The service enables governance, compliance, operational auditing, and risk auditing of users' AWS account. With the service, users can log, continuously monitor, and retain events re-

[*11] http://docs.aws.amazon.com/AWSECommerceService/latest/DG/becomingDev.html

lated to API calls across their AWS infrastructure. The service provides a history of AWS API calls for their account, including API calls made through the AWS Management Console, AWS SDKs, command line tools, and other AWS services. This history simplifies security analysis, resource change tracking, and troubleshooting[*12]. The worst point of CloudTrail for Amazon Advertising API is that the service can not log and monitor Amazon Product Advertising API at all. The service can observe other API calls and to observe all API calls can be help to notify illegal usage of AWS to users. But who can enable the monitor service that can not monitor the main use of the users? Though CloudTrail is not expensive service, it cost, is not free[*13].

AWS also provides git-secrets[*14]. This implementation prevent users from committing passwords and other sensitive information to a git repository. The implementation can easily obtain from GitHub and also install with a package manager Homebrew[*15]. Though the resolution seems to be useful, so many users do not have informed about it.

Sometimes, an integrated consolidation of APIs can cause such a lack of documentation, interface design, and collaborations with other services. Finally, AWS rejected the billing of these EC2 usage on 6 Mar. 2017. Some accused the author as the lack of thought. That is true. Meanwhile only to accuse the failure does not make any productive discussion. The author insists that sufficient amount and quality of documents may prevent such incidents because victims by a trap of Amazon Product Advertising API was not only the author.

5. Conclusion

This paper document a record concerning to the illegal use of AWS secret keys. The paper also warn a huge and prompt danger to publish the keys on public spaces such as GitHub.

参考文献

[1] Peter Mell, Timothy Grance: National Institute of Standards and Technology, U.S. Department of Commerce, Special Publication 800-145, The NIST Definition of Cloud Computing, Recommendations of the National Institute of Standards and Technology, DOI:10.6028/NIST.SP.800-145 (2011).
[2] Jason Levitt: The Web Developer's Guide To Amazon E-Commerce Service: Developing Web Applications Using Amazon Web Services And PHP, Lulu.Com, ISBN: 978-1411625518 (2005).

[*12] https://aws.amazon.com/cloudtrail/
[*13] https://aws.amazon.com/cloudtrail/pricing/
[*14] https://github.com/awslabs/git-secrets
[*15] https://brew.sh