Detection of Cookie Bomb Attacks in Cloud Computing Environment Monitored by SIEM

Ryuga Kaneko* and Taiichi Saito

Department of Information and Communication Engineering, Tokyo Denki University, Tokyo, Japan;
Email: taiichi@c.dendai.ac.jp (T.S.)
*Correspondence: 19nc016@ms.dendai.ac.jp (R.K.)

***Abstract*—This paper proposes a new method to detect Cookie Bomb attacks. A Cookie Bomb attack is a denial-of-service attack such that a user cannot receive a legitimate Hypertext Transfer Protocol (HTTP) response from an HTTP server because the total amount of cookies in an HTTP request exceeds the size limit accepted by the HTTP server. The new method includes our cloud architecture and detection algorithms. The cloud architecture distributes and executes a detection script, which is an implementation of the detection algorithms. This architecture uses Azure Virtual Machines, Azure Storage, Azure Automation, Azure Monitor, and Microsoft Sentinel. The virtual machines are the core components of the architecture, to which end users can connect via RDP to use their browsers. The detection script performs three tasks: obtaining paths to cookies databases generated by browsers, retrieving cookies data from a database, and comparing a threshold with the total size of all cookies a browser sends to a server. Results indicate that our proposed method 1) enables scheduled automation, 2) provides better visibility across regions, and 3) expands detection coverage for different Windows users, browsers, and browser profiles.**

***Keywords*—cybersecurity, cookie bomb attack, cloud computing, security information and event management**

I. Introduction

A *Cookie Bomb attack* is a denial-of-service attack such that a user cannot receive a legitimate Hypertext Transfer Protocol (HTTP) response from an HTTP server because the total amount of cookies in an HTTP request exceeds the size limit accepted by the HTTP server. It is difficult for a victim to notice the attack because the web application cannot interpret the error message from the server due to the attack. Even when the victim's browser displays an error message from the server, it is difficult for the victim not knowing this attack to determine from the content of the error message whether the browser is under attack or it is a server-side error.

When the victim's browser accesses Web Application Programming Interfaces (Web APIs), the difficulty of recognizing Cookie Bomb attacks further increases. In recent web applications, a client-side program communicates asynchronously with APIs in the background to provide a good user experience. It uses XMLHttpRequest and Fetch to make asynchronous calls to APIs. However, if the HTTP request to an Application Programming Interface (API) is poisoned with a Cookie Bomb, the server providing the API will return an error message. In the client-side program, XMLHttpRequest or Fetch receives error messages yet cannot distinguish whether the API is broken or the browser is under Cookie Bomb attacks. In RESTful APIs designs, a path parameter is used to identify a specific resource or resources instead of query parameters. An attacker can also control the Path attribute when setting the cookie. This allows the Cookie Bomb to be sent only when the victim tries to use a specific resource on the server. If a path parameter includes a user id, the Cookie Bomb is allowed to be sent only when a specific user tries to access the server. A Cookie Bomb attack can disable only the API of a specific resource by controlling the Path attribute. Thus, in a Cookie Bomb attack, an attacker can specify the target in such detail that the attack occurs only in specific cases, making it difficult for the victim to notice the attack.

To address this problem, Okazawa [1, 2] developed a browser extension for Google Chrome that detects Cookie Bomb attacks on the victim's side. It works as follows: when the browser attempts to send an HTTP request, the extension checks if the HTTP request has a cookie header using webRequest API, (if present) extracts the cookie from the header, and calculates the total cookie size. If that total exceeds a certain threshold, the extension determines that it is under attack, notifies the user by displaying an alert in a pop-up window, and intercepts the HTTP request. Then the extension deletes all the cookies it detects.

However, the purpose of browser extensions is generally intended for individuals to customize the browsing experience and functionality. This nature of browser extensions presents three problems for organizations, especially enterprises that host virtual desktops used by their employees in a cloud computing environment. The first problem is the lack of automation. Users must manually install and activate the browser extension. This limits scalability, one of the benefits cloud computing offers. The second problem is the lack of visibility. As detection results are siloed in a browser profile, enterprises cannot keep track of the attacks in a

Manuscript received July 6, 2022; revised August 2, 2022; accepted August 22, 2022; published March 14, 2023.

Security Information and Event Management (SIEM) tool that presents gathered security data via a single interface [3]. This forces enterprises to take reactive rather than proactive measures. The third problem is the limitation of detection coverage. For example, the developed Google Chrome extension does not provide detection functionality to other browsers such as Microsoft Edge and Mozilla Firefox.

Therefore, the existing solution is inadequate for organizations leveraging cloud and SIEM, as it assumes individuals. Organizations will struggle with practical detection approaches to Cookie Bomb attacks.

Given the lack of automation and visibility, as well as the limitation of detection coverage in the existing solution, this study aims to propose a new method to detect Cookie Bomb attacks and evaluate its implementation.

To achieve the research aim, we have three research objectives:

1. Design cloud architecture integrated with SIEM.
2. Develop algorithms to detect large cookies.
3. Simulate Cookie Bomb attacks for evaluation.

The research objectives posed three research questions:

RQ1. Does our proposed cloud architecture enable automation?

RQ2. Does our proposed SIEM integration provide better visibility for administrators?

RQ3. Do our proposed cloud architecture and algorithms improve detection coverage?

Answers to these questions will inform the adequacy of our proposed method.

The rest of the paper is organized as follows: Section II provides relevant background information on a cookie, Cookie Bomb attack, cloud computing, and SIEM. Section III describes cloud architecture on Azure. Section IV describes detection algorithms. Section V shows the results of attack simulations. Finally, Section VI concludes the paper.

II. Background

A. Cookie

An HTTP cookie is a mechanism for maintaining and managing user status and other information between an HTTP server and a user's browser [4]. A name-value pair called a cookie is passed from the server to the browser, and the browser stores the cookie in cookiejar and, in subsequent HTTP requests, returns the cookie to the server.

When a browser sends an HTTP request to a web server, the web server returns an HTTP response with a cookie in a Set-Cookie header. The browser stores cookies passed from the server in a database (called *cookiejar*) in memory or on disk. After that, each time an HTTP request is sent to the same server (or the same domain), the cookie is set in the Cookie header in the HTTP request.

Cookie data consists of a string called "name" and a string called "value" and is placed in Set-Cookie or Cookie headers in the form <name>=<value>. In the Set-Cookie header, optional attributes such as Expires, Domain, and Path are also placed together with cookie data.

The cookie will be sent only to the specified domain and its subdomains. The Path attribute specifies a path to which the cookie will be sent, and browsers accept an arbitrary Path attribute.

A cookie can be generated, and its attributes can also be specified via *document.cookie* in JavaScript. The domain specified in the Domain attribute should be included in the domain of the server that sends the script and should not be a public suffix [5].

Recent modern browsers such as Edge (Version 103.0.1264.71), Chrome (Version 103.0.5060.134), or Firefox (Version 102.0.1) use SQLite [6] database to implement cookiejar. Appendix A, B, and C show the relation schemas of cookies tables for Edge, Chrome, and Firefox, respectively. In Chrome and Edge, a cookie value is encrypted with AES-256 in GCM mode [7, 8] and stored in the "encrypted_value" field in the cookies table, and the "value" field is empty. As shown in Fig. 1, the value in the "encrypted_value" field (hereinafter referred to as the encrypted value) consists of a 3-byte signature "v10", a 12-byte nonce, encrypted data, and a 16-byte tag. Since the GCM mode is based on the CTR mode [9], which does not require padding, the size of ciphertext is equal to that of the original plaintext. Thus, the size of the cookie value can be computed as the size of the encrypted value minus 31 bytes without decryption. In Firefox, the cookie value is stored in the "value" field without encryption. Thus, the size of the cookie value can be obtained without decryption.

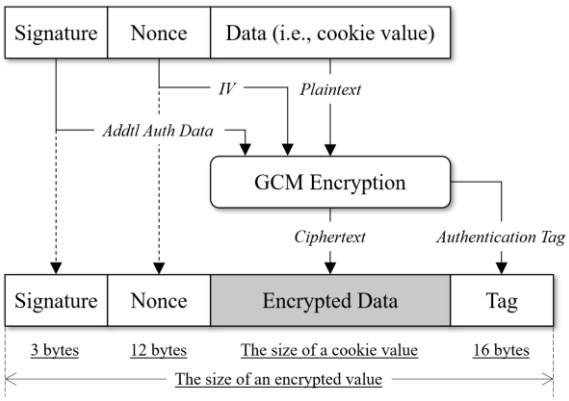


Figure 1. Using GCM to encrypt a cookie value.

B. Cookie Bomb Attack

Homakov [10] proposes a denial-of-service attack called Cookie Bomb Attack that exploits the upper limit on the size of the cookie header that server software can accept. In the attack, an attacker sets a huge size cookie, or a large number of cookies, in the victim's browser. When the browser accesses the server, it sends the cookie(s) in the cookie header. If the total amount of cookies exceeds the limit, the server returns an error message without providing the service.

Homakov also shows a proof-of-concept JavaScript snippet that, if a web page in `SUB1.example.com` includes it, causes denial of service in the entire `*.example.com` domains, including `example.com`. The snippet in a page of `SUB1.example.com` sets 99 cookies of about 4k bytes with Domain=`.example.com` in the browser. Since the snippet works in the domain `SUB1.example.com`, it can set cookies with the Domain attribute specifying a parent domain `.example.com`. Each time the browser accesses the domains `*.example.com`, these cookies are set in the cookie header of the HTTP request. Since the total amount of cookies in the HTTP request is more than 396k bytes and exceeds the limit, the error message is returned.

Relations to vulnerabilities. If a server in a sibling domain (e.g., `SUB2.example.com`) of a target domain (e.g., `target.example.com`) is vulnerable to Cross-Site Scripting (XSS) [11], an attacker can set cookies with the domain attribute equal to a common parent domain (e.g., `.example.com`) in the victim's browser, by inserting in the HTTP response a script like Homakov's snippet. Each time the victim accesses the target domain, the cookies are sent.

Similarly, if the server of a sibling domain of a target domain has a Subdomain Takeover [12] vulnerability, an attacker can take over the sibling domain and publish a page in which a script generates a cookie with the domain attribute set to a common parent domain.

If the server has an HTTP header injection vulnerability, an attacker may cause a Cookie Bomb attack by inserting a Set-Cookie header with a huge cookie into the response and setting that cookie in the browser.

Watanabe *et al*. [13] discuss an advanced attack that uses a cookie bomb attack as a component.

Cookie header size limits. A Cookie Bomb attack occurs when the total amount of cookies exceeds the size limit accepted by the server. The limit depends on the setting of the server software. The default setting of the size limit of the HTTP request header for Apache HTTP Server is 8190 bytes [14]. The default setting of the size of buffers used for reading the HTTP request header for Nginx is 8000 bytes [15]. If the limit is exceeded, the server returns a 4xx or 5xx error.

C. Cloud Computing

Cloud computing offers a variety of benefits, including scalability and manageability [16]. The National Institute of Standards and Technology (NIST) defines cloud computing as "a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction" [17].

NIST also defines three service models (i.e., Software as a Service, Platform as a Service, and Infrastructure as a Service) and four deployment models (i.e., private, community, public, and hybrid) [17].

One of the public cloud providers, Microsoft Azure, provides Azure Virtual Machines, which is an Infrastructure as a Service offering.

D. SIEM

SIEM plays a critical role in a security operations center that monitors events related to its organization's IT assets by collecting, normalizing, and analyzing security logs from diverse sources [18].

In [19], NIST shows the following two approaches for collecting logs from source hosts to a SIEM server. The agent-based approach requires an agent to be installed on a host; the agent performs filtering, collection, and normalization and then transfers logs to a centralized SIEM server. In the agentless approach, either the hosts push logs to the server or the server pulls logs from the hosts; the SIEM server then performs filtering, collection, and normalization. One of the SIEM products, Microsoft Sentinel, offers four different agents for Windows virtual machines: Azure Monitor agent, Diagnostics extension, Log Analytics agent, and Dependency agent [20].

Log generating sources are not limited to network or endpoint devices. The European Union Agency for Cybersecurity (also known as ENISA) [21] mentions that honeypots act as feeds for SIEM tools; Ailianos [22] and Yüksel [23] present the results of their implementation. Furthermore, Sankar *et al.* [24] present social media monitoring using SIEM.

III. Cloud Architecture

We propose cloud architecture, which uses Azure Virtual Machines, Azure Storage, Azure Automation, Azure Monitor, and Microsoft Sentinel. All except Sentinel belong to a resource group named "test1". The virtual machines are the core components of the architecture, to which end users can connect via RDP to use their browsers. Cloud administrators can perform vertical and horizontal scaling from the Microsoft Azure Portal. In other words, the size and number of virtual machines in the resource group can be changed later. We added four components to detect Cookie Bomb attacks on all running virtual machines in the resource group.

The first component is Azure Storage, which stores a detection script. Azure Storage offers several types of storage accounts and storage services [25]. We selected Standard general-purpose v2 as the storage account type and used Blob Storage as the storage service. Blob Storage has containers that can store multiple blobs [26]. We created a container named "vm-script-container" in which we stored a detection script named "cookieBombDetector.ps1" as a block blob.

The second component is Azure Automation, which distributes the detection script to the running virtual machines and then executes them. To get started with Azure Automation, we created an Automation Account named "test1-automation". This account allows us to isolate and manage Runbooks that can be used for task automation [27]. Azure Automation supports several types of Runbooks [28]. We created a Runbook named "test1-multiple" with PowerShell as runbook type and 7.1 as runtime version. Code 1 shows the implementation of the Runbook written in PowerShell.

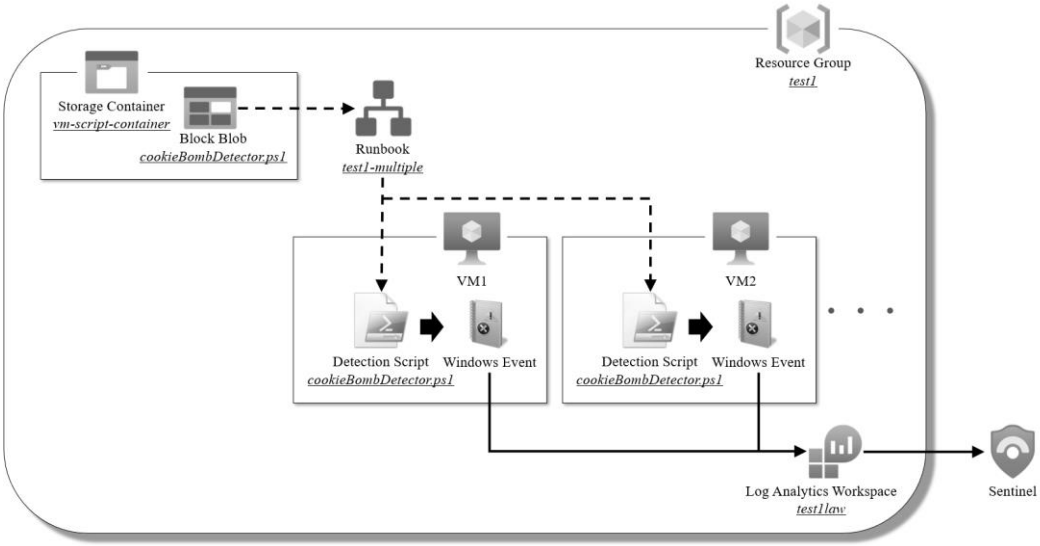


Figure 2. Proposed cloud architecture.

Code 1: Distribute and execute the detection script

```
$servicePrincipalConnection = Get-AutomationConnection -Name “AzureRunAsConnection”
Add-AzAccount -ServicePrincipal
-TenantId $servicePrincipalConnection.TenantId
-ApplicationId $servicePrincipalConnection.ApplicationId
-CertificateThumbprint $servicePrincipalConnection.CertificateThumbprint
$storageAccountKey = # Key
$azStorage = New-AzStorageContext -StorageAccountName “test1runbookstorage”
-StorageAccountKey $storageAccountKey
$blobContent = Get-AzStorageBlobContent -Container “vm-script-container”
-Blob “cookieBombDetector.ps1” -Destination ($Env:temp+”/cookieBombDetector.ps1”)
-Context $azStorage.Context -Force
$subscriptionId = # Id
Set-AzContext -Subscription $subscriptionId
$targetVMs = Get-AzVM -ResourceGroupName “test1” -status
| Where-Object {$_.PowerState -eq “VM running” -and $_.StorageProfile.OSDisk.OSType -eq “Windows”}
$targetVMs | ForEach-Object -Parallel {
    Invoke-AzVMRunCommand -ResourceGroupName “test1” -VMName $_.Name
    -CommandId “RunPowerShellScript” -ScriptPath ($Env:temp+”/cookieBombDetector.ps1”)
}
```

There are multiple methods to start the created Runbook [29]. Among them, we chose the Schedule method. This method makes it possible to automatically start the Runbook hourly, daily, weekly, or monthly. When a Runbook starts, it creates an execution instance referred to as a job [30].

The third component is Azure Monitor, which collects and stores Windows event logs. We created a Log Analytics Workspace named “test1law” to store the event logs. Log Analytics in Azure Monitor is a tool that enables the analysis of the collected logs and can be used from the Azure Portal. To collect event logs, we use the agent-based approach described in Section II-D. Log Analytics agents on Windows virtual machines can collect Windows Event logs and store them as records in the Log Analytics Workspace [31]. To deploy an Agent on a virtual machine, the virtual machine must be connected to the Log Analytics workspace. We deploy the Log Analytics Agent with the data source set to Application log to all virtual machines in the resource group “test1”.

The fourth component is Microsoft Sentinel, which generates alerts and creates incidents when a Cookie Bomb attack is detected. It is built on a Log Analytics Workspace, and we selected “test1law” as the workspace. The Microsoft Sentinel dashboard in the Azure portal provides a bird’s-eye view for attack detection. The dashboard has an analytics rule wizard [32]. With this, we can define the conditions under which Sentinel will generate an alert and create an incident simultaneously.

In this architecture, end-users do not need to perform any operations to detect Cookie Bomb attacks. The detection script is automatically distributed and executed. A cloud administrator can monitor the detection results on Sentinel without additional operations, even if the size of a virtual machine changes or the number of virtual machines increases. Fig. 2 illustrates the proposed cloud architecture.

IV. Detection Algorithm

We propose detection algorithms for Cookie Bomb attacks. Algorithm 1 detects Cookie Bomb attacks by monitoring the browser-generated cookies databases on a Windows machine.

Algorithm 1: Detect Cookie Bombs

```
1:  Initialize cookieBombs to an empty list
2:  threshold = 8000
3:  users = all enabled local users of a Windows machine
4:  For each user in users
5:      dbPaths = all paths of cookies databases under the user's
        directory
6:      For each dbPath in dbPaths
7:          cookies = cookies retrieved by running an SQL query
            to a database located at dbPath
8:          Initialize checkedUrls to an empty list
9:          For each cookie in cookies
10:             urlDomains = {cookie.host}
11:             urlPath = cookie.path
12:             If cookie.host starts with a dot
13:                 Add all subdomains of cookie.host that exist
                    in cookies to urlDomains
14:             For each urlDomain in urlDomains
15:                 url = urlDomain + urlPath
16:                 If url exists in checkedUrls
17:                     Continue
18:                 Else
19:                     Add url to checkedUrls
20:                 totalSize = CalculateTotalSize(cookies,
                    urlDomain, urlPath) // Algorithm 2
21:                 If totalSize is greater than threshold
22:                     Add {dbPath, url, totalSize} to
                        cookieBombs
23: Output cookieBombs to Windows event logs
```

The algorithm consists of three steps: 1) obtaining paths to cookies databases generated by browsers, 2) retrieving cookies data from a database, and 3) comparing a threshold with the total size of all cookies a browser sends to a server. The remainder of this section contains a detailed description of each step.

The first step is to retrieve paths to all the cookies databases, which are SQLite files generated by browsers. The retrieved paths depend on three factors: Windows user, browser, and browser profile. A single Windows machine can have multiple local users. The *users* variable stores the names of all enabled local user accounts. If an account is enabled, a user can log on and use any browser; if an account is disabled, a user cannot log on. Local users can install browsers such as Chrome and Firefox other than the pre-installed Edge. In addition, local users can create multiple profiles for each browser. The *dbPaths* variable stores the paths to the multiple cookies databases these browsers generate for each profile.

The second step is to retrieve cookies data from a database. There are two things to note about cookies databases generated by the various browsers. First, column names vary among browsers. Second, some browsers store raw cookie values while others store encrypted ones. These differences complicate the comprehensive detection of Cookie Bomb attacks on a single Windows machine with multiple browsers installed. Therefore, we created an Structured query language (SQL) query for each browser to traverse stored cookies in the same way for any browser. The resulting *cookies* variable refers to an array of objects that store both domain and path attributes of cookies and the total size of cookies. Code 2 retrieves cookies from a database generated by Edge or Chrome. Code 3 retrieves cookies from a database generated by Firefox.

Code 2: Edge or Chrome

```
SELECT host_key AS host, path,
SUM(LENGTH(name)+LENGTH(encrypted_value)-31) AS size
FROM cookies GROUP BY host_key, path
```

Code 3: Firefox

```
SELECT host, path,
SUM(LENGTH(name)+LENGTH(value)) AS size
FROM moz_cookies GROUP BY host, path
```

The third step is to compare a threshold with the total size of all cookies a browser sends to a server. Here, we specify the *threshold* as 8000. This is slightly less than the default maximum HTTP request header size for Apache HTTP Server described in Section II-B. When a browser sends a cookie, four elements play an important role. Two of them are a URL-domain and URL-path, which specify the destination of the cookie. The other two are a cookie-domain and cookie-path, attributes of the cookie that the browser will send. Here, the cookie-domain is the URL-domain or one of its parent domains, and the cookie-path is the URL-path or one of its parent paths. In other words, the browser can send multiple cookies, each with its cookie-domain and cookie-path, to a single URL consisting of the URL-domain and URL-path.

The algorithm branches depending on whether a cookie-domain, referenced as *cookie.host* at line 12 of Algorithm 1, starts with a dot. If a cookie-domain does not start with a dot, the selected *cookie* is visible only from the cookie-domain. Thus, a URL-domain will be the same as the cookie-domain. For example, a cookie set to `example.com` will only be sent to `example.com`, not to `SUB1.example.com`. If a cookie-domain starts with a dot, the selected *cookie* will be visible to the cookie-domain and all its subdomains. Thus, the *urlDomains* variable stores the cookie-domain and all its subdomains. For example, a cookie set to `.example.com` will be sent to `example.com` and its subdomains, such as `SUB1.example.com`.

The CalculateTotalSize function calculates the total size of all cookies that should be sent to a URL consisting of a URL-domain and URL-Path arguments, as shown in Algorithm 2.

Algorithm 2: Calculate the total size of visible cookies

```
Function CalculateTotalSize(cookies, urlDomain, urlPath)
    cDomains = GetCookieDomains(urlDomain) //Algorithm3
    cPaths = GetCookiePaths(urlPath) // Algorithm 4
    Initialize totalSize to 0
    For each cDomain in cDomains
        For each cPath in cPaths
            visibleCookie = a cookie with cDomain and cPath
            in cookies
            If visibleCookie is not null
                totalSize += the size of visibleCookie
    Return totalSize
```

Algorithm 2 uses two other defined functions: Algorithm 3 and Algorithm 4.

Algorithm 3: Get cookie domains

```
Function GetCookieDomains(urlDomain)
    Initialize cDomains to an empty list
    Add urlDomain to cDomains
    If urlDomain does not start with a dot
        Add a dot-prefixed urlDomain to cDomains
    dotIndex = leftmost index of a dot in urlDomain with a start index of 1 (index is zero-based)
    While dotIndex is greater than 0
        Add a substring of urlDomain with a start index of dotIndex to cDomains
        dotIndex = leftmost index of a dot in urlDomain with a start index of dotIndex +1, or −1 if not found
    Return cDomains
```

Algorithm 4: Get cookie paths

```
Function GetCookiePaths(urlPath)
    Initialize cPaths to an empty list
    Add urlPath to cPaths
    If urlPath is not a slash
        slashIndex = rightmost index of a slash in urlPath (index is zero-based)
        While slashIndex is greater than 0
            Add a substring of urlPath with a start index of 0 and end index of slashIndex to cPaths
            slashIndex = rightmost index of a slash in urlPath with a start index of slashIndex −1, or −1 if not found
        Add a slash to cPaths
    Return cPaths
```

V. Evaluation

A. Cloud Architecture

To evaluate the effectiveness of the proposed cloud architecture against Cookie Bomb attacks, we conducted an experiment.

Experimental setup. First, we created three virtual machines in the resource group: VM-JapanEast, VM-EastUS, and VM-UKSouth. They all share the same size and operating system: Standard B2ms (2 vcpus, 8 GiB memory) and Windows 10, respectively. A word after a hyphen found in these virtual machines' names corresponds to a deployment region, and Windows clocks are set to local time.

Second, we installed XAMPP [33], an Apache distribution, on all virtual machines. Configuration 1 shows name-based virtual host configurations [34] that we added to use multiple domains on a single Apache HTTP server. Configuration 2 shows entries we added to the Windows hosts file so that domain names map to IP addresses.

Configuration 1: VirtualHost

```
<VirtualHost *:80>
    ServerName example.com
    DocumentRoot "C:\xampp\htdocs\example.com"
</VirtualHost>
<VirtualHost *:80>
    ServerName sub1.example.com
    DocumentRoot "C:\xampp\htdocs\sub1.example.com"
</VirtualHost>
```

Configuration 2: hosts

```
127.0.0.1 example.com
127.0.0.1 sub1.example.com
```

Third, we connected these virtual machines to the Log Analytics Workspace.

Fourth, we set Microsoft Sentinel to query stored event logs every 10 minutes. Code 4 shows the query written in Kusto Query Language (KQL) that searches for events indicating detection of Cookie Bomb attacks.

Code 4: Search for events indicating detection of the attacks

```
Event
| where * contains "Cookie Bomb detected at"
| project TimeGenerated, Computer, RenderedDescription
| order by TimeGenerated
```

Fifth, we set the Runbook to automatically distribute and execute the detection script daily at 14:00 and 16:00. Unless we explicitly state otherwise, the time is in Japan Standard Time (UTC+9) throughout this section.

Attack simulation. Table I shows an attack scenario targeting all the virtual machines in the resource group. Browsers whose Cookie Bomb column has an asterisk are subjected to Cookie Bomb attacks. This attack scenario consists of two steps. Step 1 starts with the power status of all virtual machines in the "Stopped" state. At 13:00, we started VM-JapanEast and performed a Cookie Bomb attack against its Edge. Step 2 starts with VM-JapanEast's power status set to "Running". At 15:00, we started both VM-EastUS and VM-UKSouth. Shortly after, we attacked VM-JapanEast and VM-EastUS out of the three running virtual machines.

TABLE I. Attack Scenario

Virtual Machine	Windows User	Browser	Browser Profile	Cookie Bomb	ID
VM-JapanEast	JapanEastUser1	Edge	Profile 1	*	1
VM-EastUS	EastUSUser1	Edge	Profile 1	*	2
		Edge	Profile 2	*	3
		Chrome	Profile 1	*	4
		Firefox	Profile 1	*	5
	EastUSUser2	Edge	Profile 1	*	6
VM-UKSouth	UKSouthUser1	Edge	Profile 1		7

Job history of the Runbook. Fig. 3 shows a job history of the Runbook. We can see the Runbook began distributing and executing the detection script at 14:00:06 during step 1 and at 16:00:03 during step 2. This history indicates that the Runbook was successfully scheduled as we set. Our answer to RQ1. *Does our proposed cloud architecture enable automation?* is that the Runbook in the cloud architecture enables scheduled automation.

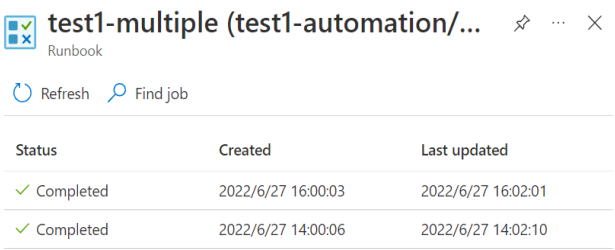


Figure 3. Job history of the Runbook.

TABLE II. SUMMARY OF WINDOWS EVENT LOGS

Windows Clock	Virtual Machine	Event Message
2:00:48 PM	VM-JapanEast	Cookie Bomb detected at: C:\Users\JapanEastUser1\AppData\Local\Microsoft\Edge\User Data \Default\Network\Cookies example.com / 9033
4:01:08 PM	VM-JapanEast	Cookie Bomb detected at: C:\Users\JapanEastUser1\AppData\Local\Microsoft\Edge\User Data \Default\Network\Cookies example.com / 9033
3:01:27 AM	VM-EastUS	Cookie Bomb detected at: C:\Users\EastUSUser1\AppData\Local\Microsoft\Edge\User Data \Default\Network\Cookies example.com / 9033 Cookie Bomb detected at: C:\Users\EastUSUser1\AppData\Local\Microsoft\Edge\User Data \Profile 1\Network\Cookies example.com / 9033 Cookie Bomb detected at: C:\Users\EastUSUser1\AppData\Local\Google\Chrome\User Data \Default\Network\Cookies example.com / 9033 Cookie Bomb detected at: C:\Users\EastUSUser1\AppData\Roaming\Mozilla\Firefox\Profiles \o5hzcq40.default-release\cookies.sqlite example.com / 9033 Cookie Bomb detected at: C:\Users\EastUSUser2\AppData\Local\Microsoft\Edge\User Data \Default\Network\Cookies example.com / 9033
8:01:24 AM	VM-UKSouth	Cookie Bomb not detected.

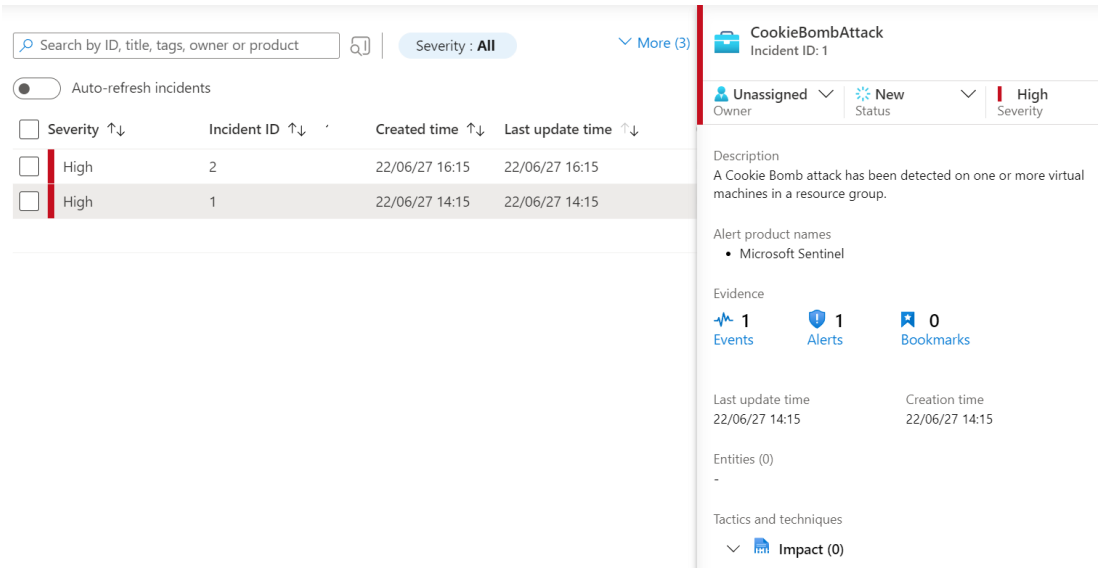


Figure 4. Incident page of Microsoft Sentinel.

Windows event logs. A detection script distributed on a virtual machine outputs results in the form of event logs. Table II shows a summary of Windows event logs from the three virtual machines we created. Each event message contains the output of the detection script. The first event log was generated on VM-JapanEast at 14:00:48 during step 1. The absolute path to the Cookies database reveals a victim; the user "JapanEastUser1" used Edge with the default profile. This victim matches the row with the ID of 1 in Table I. The second event log was generated on VM-JapanEast at 16:01:08 during step 2. The script detected the attack again as we did not remove the Cookie Bomb set during step 1. The third event log was generated on VM-EastUS at 03:01:27 (UTC-4) during step 2. We should note that the time is in Eastern Daylight Time since we set Windows clocks to local time. Victims revealed from the absolute file paths match the rows with IDs of 2, 3, 4, 5, and 6 in Table I. Finally, the fourth event log was generated on VM-UKSouth at 08:01:24 (UTC+1) during step 2. Similar to the third log, note that the time is in British Summer Time. This time, the event message is "Cookie Bomb not detected," which matches the row with the ID of 7 in Table I. These relations between each event log in Table II and a row in Table I indicate successful detection of Cookie Bomb attacks.

Microsoft Sentinel. As shown in Fig. 4, Sentinel created two incidents in the experiment. Each incident has a related event(s) as a piece of evidence. We confirmed the following two results. First, the incident with the ID of 1 created at 14:15 has evidence linked to the first event log in Table II. Second, the incident with the ID of 2 created at 16:15 has pieces of evidence linked to the second and third event logs in Table II. These two results indicate successful aggregation of events detected Cookie Bomb attacks. It should be emphasized that the time on the incident page is in Japan Standard Time (UTC+9), while some events of evidence were generated in different time zones. This time conversion enables us to track the attack scenario chronologically. Our answer to RQ2. *Does our proposed SIEM integration provide better visibility for administrators?* is that Sentinel provides better visibility across regions.

B. *Detection Algorithms*

To evaluate the correctness of the proposed detection algorithms against Cookie Bomb attacks, we conducted an experiment.

Experimental setup. We used EastUSUser1 of VM-EastUS created in the previous experiment. The detection script can run independently without being invoked by the Runbook.

Experiment. We simulated four cases of Cookie Bomb attacks. In either case, Edge, Chrome, and Firefox each use their default profile to load a page containing an attack script that sets a cookie. To simplify evaluation, we cleared existing cookies in browsers before loading pages in each case. We evaluate the correctness of the algorithms for each simulated attack case by showing the followings: Code, Table, and Output.

First, the data denoted Code in each case is an implementation of an attack script. From the *document.cookie* property, we can calculate the total size of the cookies to be set.

Second, the data denoted table in each case is the contents of a *cookies* variable declared in Algorithm 1. The detection script will initialize a *cookies* variable three times for Edge, Chrome, and Firefox. We want to emphasize that those variables will be identical since all browsers open the same page after all the existing cookies are deleted. The table has three columns: host, path, and size. If the value in the size column matches our calculated value, it indicates the correctness of calculations done by our Structured Query Language (SQL) queries shown in Code 2 and Code 3.

Third, the data denoted Output in each case is detection results, including three pieces of information. The first one is an absolute path to the database, which contains cookies we set. As mentioned in the previous subsection, this absolute path reveals a victim's browser. If the path includes the name of a browser, it indicates that detection was successful for that browser. The second one is a set of a domain and a path indicating pages affected by the attack. Finally, the third one is the total size of cookies that a browser could send to the affected pages. Suppose the output has results of Edge, Chrome, and Firefox, and its sizes equal our calculated value. In that case, it indicates the successful detection of Cookie Bomb attacks for all the browsers.

Case 1. The first case is when the server of `example.com` sets three cookies of 3003 bytes each in browsers, as shown in Code 5. From the code, the total size of the cookies to be set is (3 bytes of name + 3000 bytes of value) × 3 cookies = 9009 bytes. Table III shows the *cookies* variable, and Output 1 shows the detection result. We can confirm the successful detection of the Cookie Bomb attacks for all the browsers from the three data types.

Code 5: Attack script for case 1

```
for (var i = 1; i <= 3; i++) {
    name = i.toString().padStart(3, 0);
    value = 'a'.padEnd(3000, 'a');
    document.cookie = name + '=' + value +
    '; max-age=31536000';
}
```

TABLE III. *COOKIES* VARIABLE IN CASE 1

host	path	size
example.com	/	9009

Output 1: Detection result of case 1

```
Cookie Bomb detected at: C:\Users\EastUSUser1\AppData
\Local\Microsoft\Edge\User Data\Default\Network\Cookies
example.com / 9009
Cookie Bomb detected at: C:\Users\EastUSUser1\AppData
\Local\Google\Chrome\User Data\Default\Network\Cookies
example.com / 9009
Cookie Bomb detected at: C:\Users\EastUSUser1\AppData
\Roaming\Mozilla\Firefox\Profiles\o5hzcq40.default-release
\cookies.sqlite   example.com / 9009
```

Case 2. The second case is when the server of `example.com` sets a hundred cookies of 103 bytes each in browsers, as shown in Code 6. From the code, the total size of the cookies to be set is (3 bytes of name + 100 bytes of value) × 100 cookies = 10300 bytes. Table IV shows the *cookies* variable, and Output 2 shows the detection result. We can confirm the successful detection of the Cookie Bomb attacks for all the browsers from the three data types.

Code 6: Attack script for case 2

```
for (var i = 1; i <= 100; i++) {
    name = i.toString().padStart(3, 0);
    value = 'a'.padEnd(100, 'a');
    document.cookie = name + '=' + value +
    '; domain=example.com; max-age=31536000';
}
```

TABLE IV. *COOKIES* VARIABLE IN CASE 2

host	path	size
.example.com	/	10300

Output 2: Detection result of case 2

```
Cookie Bomb detected at: C:\Users\EastUSUser1\AppData
\Local\Microsoft\Edge\User Data\Default\Network\Cookies
.example.com / 10300
Cookie Bomb detected at: C:\Users\EastUSUser1\AppData
\Local\Google\Chrome\User Data\Default\Network\Cookies
.example.com / 10300
Cookie Bomb detected at: C:\Users\EastUSUser1\AppData
\Roaming\Mozilla\Firefox\Profiles\o5hzcq40.default-release
\cookies.sqlite   .example.com / 10300
```

Case 3. The third case is when the server of `sub1.example.com` sets nine cookies, each with different property, as shown in Code 7. From the code, the total size of the cookies to be set is (3 bytes of name + 1000 bytes of value) × 9 cookies = 9027 bytes. Table V shows the cookies variable, and Output 3 shows the detection result. We can confirm the successful detection of the Cookie Bomb attacks for all the browsers from the three data types.

Code 7: Attack script for case 3

```
value = 'a'.padEnd(1000, 'a');
document.cookie = '001=' + value +
'; domain=example.com; max-age=31536000';
document.cookie = '002=' + value +
'; domain=example.com; path=/sub1; max-age=31536000';
document.cookie = '003=' + value +
'; domain=example.com; path=/sub1/sub2; max-
age=31536000';
document.cookie = '004=' + value +
'; domain=sub1.example.com; max-age=31536000';
document.cookie = '005=' + value +
'; domain=sub1.example.com; path=/sub1; max-
age=31536000';
document.cookie = '006=' + value +
```

```
    '; domain=sub1.example.com; path=/sub1/sub2; max-
    age=31536000';
8:  document.cookie = '007=' + value +
    '; max-age=31536000';
9:  document.cookie = '008=' + value +
    '; path=/sub1; max-age=31536000';
10: document.cookie = '009=' + value +
    '; path=/sub1/sub2; max-age=31536000';
```

TABLE V. *COOKIES* VARIABLE IN CASE 3

host	path	size
.example.com	/	1003
.example.com	/sub1	1003
.example.com	/sub1/sub2	1003
.sub1.example.com	/	1003
.sub1.example.com	/sub1	1003
.sub1.example.com	/sub1/sub2	1003
sub1.example.com	/	1003
sub1.example.com	/sub1	1003
sub1.example.com	/sub1/sub2	1003

Output 3: Detection result of case 3

```
1: Cookie Bomb detected at: C:\Users\EastUSUser1\AppData
   \Local\Microsoft\Edge\User Data\Default\Network\Cookies
   sub1.example.com /sub1/sub2 9027
2: Cookie Bomb detected at: C:\Users\EastUSUser1\AppData
   \Local\Google\Chrome\User Data\Default\Network\Cookies
   sub1.example.com /sub1/sub2 9027
3: Cookie Bomb detected at: C:\Users\EastUSUser1\AppData
   \Roaming\Mozilla\Firefox\Profiles\o5hzcq40.default-release
   \cookies.sqlite   sub1.example.com /sub1/sub2 9027
```

Case 4. The fourth case is when the server of `sub1.example.com` sets five cookies, each with different property, as shown in Code 8. From the code, the total size of the cookies to be set is (3 bytes of name + 1800 bytes of value) × 5 cookies = 9015 bytes. Table VI shows the *cookies* variable, and Output 4 shows the detection result. We can confirm the successful detection of the Cookie Bomb attacks for all the browsers from the three data types.

Here we are interested in the relation: Output 4 shows the affected page `.sub1.example.com/sub1/sub2` while Table VI does not have any corresponding cookies. As we described, the existing Chrome extension detects Cookie Bomb attacks by checking an HTTP request with webRequest API, and then requires users to visit affected pages in order to detect attacks. On the other hand, the relation indicates that our proposed algorithms can detect possible Cookie Bomb attacks even before users visit the pages. Therefore, the use of cookies databases in the detection process provides better detection results over webRequest API.

Code 8: Attack script for case 4

```
1: value = 'a'.padEnd(1800, 'a');
2: document.cookie = '001=' + value +
   '; domain=example.com; max-age=31536000';
3: document.cookie = '002=' + value +
   '; domain=example.com; path=/sub1; max-age=31536000';
4: document.cookie = '003=' + value +
   '; domain=example.com; path=/sub1/sub2; max-
   age=31536000';
5: document.cookie = '004=' + value +
   '; domain=sub1.example.com; max-age=31536000';
6: document.cookie = '005=' + value +
   '; domain=sub1.example.com; path=/sub1; max-
   age=31536000';
```

TABLE VI. *COOKIES* VARIABLE IN CASE 4

host	path	size
.example.com	/	1803
.example.com	/sub1	1803
.example.com	/sub1/sub2	1803
.sub1.example.com	/	1803
.sub1.example.com	/sub1	1803

Output 4: Detection result of case 4

```
1: Cookie Bomb detected at: C:\Users\EastUSUser1\AppData
   \Local\Microsoft\Edge\User Data\Default\Network\Cookies
   .sub1.example.com /sub1/sub2 9015
2: Cookie Bomb detected at: C:\Users\EastUSUser1\AppData
   \Local\Google\Chrome\User Data\Default\Network\Cookies
   .sub1.example.com /sub1/sub2 9015
3: Cookie Bomb detected at: C:\Users\EastUSUser1\AppData
   \Roaming\Mozilla\Firefox\Profiles\o5hzcq40.default-release
   \cookies.sqlite   .sub1.example.com /sub1/sub2 9015
```

Our answer to RQ3. *Do our proposed cloud architecture and algorithms improve detection coverage?* is that they expand detection coverage for different Windows users, browsers, and browser profiles.

VI. CONCLUSION

In this section, we first revisit the research aim. Then, we discuss the limitations of this study.

This study aimed to propose a new method to detect Cookie Bomb attacks and evaluate its implementation. The results indicate that our proposed method: 1) enables scheduled automation, 2) provides better visibility across regions, and 3) expands detection coverage for different Windows users, browsers, and browser profiles.

During our evaluation, we used only a few virtual machines to reduce cloud costs. Consequently, we could not verify the scalability of the method when the number of virtual machines increases significantly. We are also aware that there has been no evaluation of the algorithm's execution time.

CONFLICT OF INTEREST

The authors declare no conflict of interest.

AUTHOR CONTRIBUTIONS

Ryuga Kaneko conducted the research and wrote the paper. Taiichi Saito supervised the research and the paper. All authors had approved the final version.

ACKNOWLEDGMENT

The authors would like to thank the reviewers of the 15th International Conference on Computer Science and Information Technology (ICCSIT 2022) for their comments and suggestions.

APPENDIX A: RELATION SCHEMA OF COOKIES TABLE FOR EDGE

```
CREATE TABLE cookies(
    creation_utc INTEGER NOT NULL,
    host_key TEXT NOT NULL,
    name TEXT NOT NULL,
    value TEXT NOT NULL,
```

```
    path TEXT NOT NULL,
    expires_utc
    INTEGER NOT NULL,
    is_secure INTEGER NOT NULL,
    is_httponly INTEGER NOT NULL,
    last_access_utc INTEGER NOT NULL,
    has_expires INTEGER NOT NULL DEFAULT 1,
    is_persistent INTEGER NOT NULL DEFAULT 1,
    priority INTEGER NOT NULL DEFAULT 1,
    encrypted_value BLOB DEFAULT ‘‘,
    samesite INTEGER NOT NULL DEFAULT -1,
    source_scheme INTEGER NOT NULL DEFAULT 0,
    is_edgelegacycookie INTEGER DEFAULT 0,
    browser_provenance INTEGER DEFAULT 0,
    UNIQUE (host_key, name, path)
)
```

APPENDIX B: RELATION SCHEMA OF COOKIES TABLE FOR CHROME

```
CREATE TABLE cookies(
    creation_utc INTEGER NOT NULL,
    host_key TEXT NOT NULL,
    top_frame_site_key TEXT NOT NULL,
    name TEXT NOT NULL,
    value TEXT NOT NULL,
    encrypted_value BLOB NOT NULL,
    path TEXT NOT NULL,
    expires_utc INTEGER NOT NULL,
    is_secure INTEGER NOT NULL,
    is_httponly INTEGER NOT NULL,
    last_access_utc INTEGER NOT NULL,
    has_expires INTEGER NOT NULL,
    is_persistent INTEGER NOT NULL,
    priority INTEGER NOT NULL,
    samesite INTEGER NOT NULL,
    source_scheme INTEGER NOT NULL,
    source_port INTEGER NOT NULL,
    is_same_party INTEGER NOT NULL
)
```

APPENDIX C: RELATION SCHEMA OF COOKIES TABLE FOR FIREFOX

```
CREATE TABLE moz_cookies (
    id INTEGER PRIMARY KEY,
    originAttributes TEXT NOT NULL DEFAULT ‘‘,
    name TEXT,
    value TEXT,
    host TEXT,
    path TEXT,
    expiry INTEGER,
    lastAccessed INTEGER,
    creationTime INTEGER,
    isSecure INTEGER,
    isHttpOnly INTEGER,
    inBrowserElement INTEGER DEFAULT 0,
    sameSite INTEGER DEFAULT 0,
    rawSameSite INTEGER DEFAULT 0,
    schemeMap INTEGER DEFAULT 0,
    CONSTRAINT moz_uniqueid
    UNIQUE (name, host, path, originAttributes)
)
```

REFERENCES

[1] Y. Okazawa and T. Saito, “Browser extension to detect cookie bomb attack,” (in Japanese), in *Proc. the 39th Symposium on Cryptography and Information Security*, 2022.

[2] Y. Okazawa, “Browser extension to detect cookie bomb attack,” (in Japanese), M. S. thesis, Graduate School of Engineering, Tokyo Denki University, 2022.

[3] A. Johnson *et al*., “Guide for security-focused configuration management of information systems,” National Institute of Standards and Technology Special Publication, 2011. doi: 10.6028/NIST.SP.800-128.

[4] A. Barth. (2011). RFC 6265: HTTP state management mechanism. IETF RFC6265. [Online]. Available: https://www.rfc-editor.org/rfc/rfc6265

[5] Mozilla Foundation. Public Suffix List. [Online]. Available: https://publicsuffix.org/

[6] R. D. Hipp. (2020). SQLite. [Online]. Available: https://www.sqlite.org/index.html

[7] M. Dworkin, “Recommendation for block cipher modes of operation: Galois/Counter Mode (GCM) and GMAC,” National Institute of Standards and Technology Special Publication, 2007. doi: 10.6028/NIST.SP.800-38D.

[8] D. A. McGrew and J. Viega. The Galois/Counter Mode of Operation (GCM). [Online]. Available: https://csrc.nist.rip/groups/ST/toolkit/BCM/documents/proposedmodes/gcm/gcm-spec.pdf

[9] International Organization for Standardization. ISO/IEC 10116:2006. Information technology — Security techniques — Modes of operation for an n-bit block cipher. (2006). [Online]. Available: https://www.iso.org/standard/50476.html

[10] E. Homakov, “Cookie bomb or let’s break the internet,” [Online]. Available: https://homakov.blogspot.com/2014/01/cookie-bomb-or-lets-break-internet.html

[11] S. Kirsten. OWASP Foundation. Cross Site Scripting (XSS) software attack. [Online]. Available: https://owasp.org/www-community/attacks/xss/

[12] HackerOne. (2018). A guide to subdomain takeovers. [Online]. Available: https://www.hackerone.com/application-security/guide-subdomain-takeovers

[13] T. Watanabe, E. Shioji, M. Akiyama, and T. Mori, “Melting pot of origins: Compromising the intermediary web services that rehost websites,” in *Proc. Network and Distributed Systems Security (NDSS) Symposium 2020*. doi: 10.14722/ndss.2020.24140

[14] Apache software foundation. LimitRequestFieldSize Directive. Apache HTTP Server Version 2.4 Documentation. [Online]. Available: https://httpd.apache.org/docs/2.4/en/mod/core.html#LimitRequestFieldSize

[15] large_client_header_buffers. In Nginx documentation. [Online]. Available: http://nginx.org/en/docs/http/ngx_http_core_module.html#large_client_header_buffers

[16] C. Harding, *Cloud Computing for Business -The Open Group Guide*, Zaltbommel, Netherlands: Van Haren Publishing, 2011.

[17] P. Mell and T. Grance, “The NIST definition of cloud computing,” National Institute of Standards and Technology Special Publication, 2011. doi: 10.6028/NIST.SP.800-145

[18] S. Bhatt, P. K. Manadhata and L. Zomlot, “The operational role of security information and event management systems,” *IEEE Security & Privacy*, vol. 12, no. 5, pp. 35–41, 2014. doi: 10.1109/MSP.2014.103.

[19] K. Kent and M. Souppaya, “Guide to computer security log management,” National Institute of Standards and Technology Special Publication, 2006. doi: 10.6028/NIST.SP.800-92

[20] Microsoft. Overview of Azure Monitor agents. [Online]. Available: https://docs.microsoft.com/en-us/azure/azure-monitor/agents/agents-overview#windows-agents

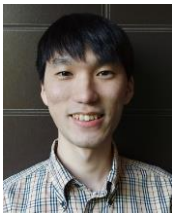
[21] European Union Agency for Cybersecurity (ENISA). (2012). Proactive detection of security incidents II – Honeypots. [Online]. Available: https://www.enisa.europa.eu/publications/proactive-detection-of-security-incidents-II-honeypots

[22] T. A. Ailianos, “SIEM optimization using honeypots,” M.S. thesis, School of Science, National and Kapodistrian University of Athens, 2014.

[23] E. Yüksel, “Experimenting, threat detection and SIEM integration with custom created honeypots,” M.S. thesis, Graduate School of Natural and Applied Sciences, Bahcesehir University, Istanbul, 2019.

[24] P. Sankar, D. E. George, and S. N. S. Aparna, “Social media monitoring using ELK Stack,” in *Proc. 2022 IEEE International Conference on Signal Processing, Informatics, Communication and Energy Systems (SPICES)*, 2022, pp. 231–235. doi: 10.1109/SPICES52834.2022.9774273

[25] Microsoft. Storage account overview. [Online]. Available: https://docs.microsoft.com/en-us/azure/storage/common/storage-account-overview#types-of-storage-accounts
[26] Microsoft. Introduction to Blob (object) storage. [Online]. Available: https://docs.microsoft.com/en-us/azure/storage/blobs/storage-blobs-introduction#containers
[27] Microsoft. Azure Automation account authentication. [Online]. Available: https://docs.microsoft.com/en-us/azure/automation/automation-security-overview#automation-account
[28] Microsoft. Azure Automation runbook types. [Online]. Available: https://docs.microsoft.com/en-us/azure/automation/automation-runbook-types
[29] Microsoft. Start a runbook in Azure Automation. [Online]. Available: https://docs.microsoft.com/en-us/azure/automation/start-runbooks
[30] Microsoft. Runbook execution in Azure Automation. [Online]. Available: https://docs.microsoft.com/en-us/azure/automation/automation-runbook-execution
[31] Microsoft. Log Analytics agent data sources in Azure Monitor. [Online]. Available: https://docs.microsoft.com/en-us/azure/azure-monitor/agents/agent-data-sources#log-records
[32] Microsoft. Create custom analytics rules to detect threats with Microsoft Sentinel. [Online]. Available: https://docs.microsoft.com/en-us/azure/sentinel/detect-threats-custom#create-a-custom-analytics-rule-with-a-scheduled-query
[33] Apache Friends. XAMPP. [Online]. Available: https://www.apachefriends.org/
[34] Apache Software Foundation. Apache Virtual Host documentation. [Online]. Available: https://httpd.apache.org/docs/2.4/vhosts/



Ryuga Kaneko was born in Yokohama, Japan and is a senior majoring in information and communication engineering at Tokyo Denki University. He is currently an undergraduate student in the Cryptography Laboratory led by Professor Taiichi Saito. His research interests include cybersecurity and cloud technology.

Taiichi Saito has a bachelor degree and a master degree of science from Waseda University, Tokyo, Japan, and a doctor degree of engineering from Chuo University, Tokyo, Japan. Since 2008, he has been a professor at Tokyo Denki University. His research work focuses on cybersecurity and software vulnerability.