Detection of Systemic Security Vulnerabilities in Cloud-Based Healthcare Infrastructure: Identifying Server-Side Request Forgery in Medical Data Systems

Dr. Sofia Almeida[1]*, Dr. Ricardo Mendes[1]
[1]Department of Health Informatics and Cybersecurity, Hospital Clínico San Carlos, Madrid, Spain

Abstract

Cloud infrastructure offers significant benefits to organizations capable of leveraging rich application programming interfaces (APIs) to automate environments at scale. However, unauthorized access to management APIs can enable threat actors to compromise the security of large amounts of sensitive data very quickly. Practitioners have documented techniques for gaining access through Server-Side Request Forgery (SSRF) vulnerabilities that exploit management APIs within cloud providers. However, mature organizations have failed to detect some of the most significant breaches, sometimes for months after a security incident. Cloud services adoption is increasing, and firms need effective methods of detecting SSRF attempts to identify threats and mitigate vulnerabilities. This paper examines a variety of tools and techniques to detect SSRF activity within an Amazon Web Services (AWS) environment that can be used to monitor for real-time SSRF exploit attempts against the AWS API. The research findings outline the efficacy of four different strategies to answer the question of whether security professionals can leverage additional vendor-provided and open-source tools to detect SSRF attacks.

1. Introduction

Cloud infrastructure-as-a-service providers have experienced tremendous growth as firms have replaced on-premises equipment with highly scalable, outsourced offerings in technology refresh cycles. Per-minute billing and workload bidding models for computing and storage make platforms like Amazon Web Services (AWS), Microsoft Azure, and Google Cloud Engine attractive for businesses looking to optimize costs and experiment without significant investments in capital expenditure for infrastructure. Pay-as-you-go access to machine learning analysis and natural language processing services attract technologists and developers alike.

Rich APIs for the management plane of cloud service providers allow for complex deployments on a massive scale with minimal manual effort. New toolchains, such as Ansible and Terraform, have emerged to implement "infrastructure as code" and track the state of globally distributed resources, underscoring both the breadth of these APIs and the growing complexity to manage them without additional technology layers. First and foremost, these services are designed to enable rapid prototyping, adoption, and deployment and employ a trust model that prioritizes automation.

Systems that are vulnerable to SSRF share a common, prolific weakness to injection attacks: insufficient input validation and improper handling permit unauthorized access or modifications of an underlying or connected system (OWASP, 2017). When a client can inject commands into a server process, which in turn reissues from the context of that process, a Server-Side Request Forgery (SSRF) vulnerability exists. An attacker can exploit an SSRF vulnerability to issue HTTP requests to internal resources, and in the case of AWS, access a sensitive internal resource called the EC2 Instance Metadata service (IMS). The IMS supports automation tools in many ways, including returning temporary credentials that attackers can leverage to access and manipulate other cloud resources through the AWS API. It is available to EC2 instances through a well-known link-local address of 169.254.169.254.

On-premises or co-located environments provide engineers with opportunities to install intrusion detection systems (IDS) that could detect injection and SSRF attacks. Historically, though, cloud services like AWS both disallowed collocation of physical

appliances in their managed infrastructure and provided no “port mirroring” or “span port” features that could leverage traditional packet capture and analysis techniques for detection. While administrators can deploy inline virtual appliances, such as inspection proxies, in AWS, scalability patterns often required multiple layers of Elastic Load Balancers to support an inline IDS strategy. Furthermore, until 2017, Elastic Load Balancers were only able to handle TCP traffic (Amazon Web Services, 2017).

However, recently available features like AWS VPC Traffic Mirroring now natively support out-of-band IDS designs in the cloud. Also, SSRF attacks often leave artifacts detectable through cloud-native threat detection products like Amazon GuardDuty, and sometimes through host facilities, such as `auditd` and `iptables`. This paper researches the efficacy of these various tools and techniques in detecting such SSRF exploitation attempts, using both freely available and cloud vendor-provided tools.

2. Research Method

Injection flaws have been well documented and widely exploited for over 20 years (rain.forest.puppy, 1998). While often described as an attack that induces the vulnerable system to dispatch an HTTP request, SSRF attacks can leverage other application layer protocols addressable with a URI or do more than gather sensitive data. Other vectors include FTP (ERPScan, 2013), XXE attacks that map the internal environment (Institute of Information Security, 2015), and variations of SSRF that achieve remote code execution when used as a channel to deliver a Shellshock payload (Kettle, 2017). Generally, SSRF requires three conditions to compromise the AWS API: the ability to inject a command, a vulnerable component that will mishandle the payload and issue a request to the AWS API, and a return path for the AWS API response to a resource the attacker can observe.

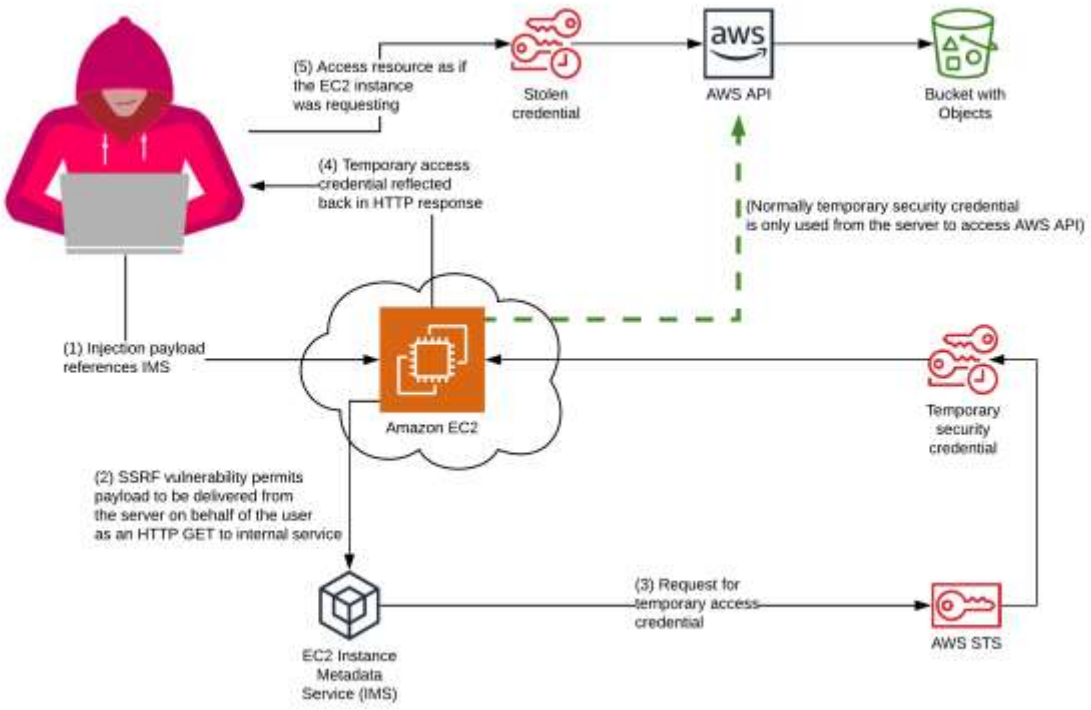

Figure 1 Overview of an SSRF attack against EC2 Instance Metadata Service to access a protected S3 bucket

An example of a vulnerable Node.js Express web application which incorporates these three required elements is provided below.

```
const request = require('request');
app.get('/avatar', function(req, response) {
    request.get(request.query['url']).pipe(response);
});
```

Figure 2 Excerpt of a Node.js Express program vulnerable to SSRF

This example would execute on a webserver, open and read the contents of a file, and return it to the caller, even if that `url` parameter were a fully formed URL to a remote system. Because an input parameter specifies the location of the file and because the method fails to perform input validation to ensure the file location is not a URL of a remote system, this vulnerability can be exploited to induce the webserver to make an HTTP GET call to a protected resource that the server, but not the attacker, can directly access.

This research examined a Node.js application vulnerable to SSRF (Art, 2016) on an AWS EC2 t3a.small instance running Amazon Linux 2. Elastic Load Balancers terminated TLS to provide plaintext analysis of traffic to the server. By sending an SSRF payload to the EC2 IMS at http://169.254.169.254/iam/security-credentials/*role-name* leveraging this technique, the external tests obtained temporary credentials from the internal instance. For each test, the researcher reviewed Amazon GuardDuty, AWS VPC Traffic Mirroring, and host-based facilities `auditd` and `iptables` to determine if each system could positively identify either the request or the response and whether findings were distinguishable from non-attack accesses of the EC2 IMS. Using the dynamic capabilities of the AWS cloud, the researcher recreated a vulnerable target host between each test to ensure observations used new temporary credentials, otherwise temporary credentials could remain the same for several hours.

The instance type t3a.small was sufficient to minimally test the four detection methods because it is part of the next generation of instances termed “AWS Nitro” that support VPC Traffic Mirroring (Amazon Web Services, 2017).

3. Findings and Discussion

3.1. AWS Attack Surface Area

AWS publishes best practices that discourage the profiling of long-lasting AWS API credentials and encourages the use of Identity and Access Management (IAM) roles applied to EC2 instances through an Instance Profile. Policies that grant permissions to AWS resources are applied to instances when they are attached to one IAM role linked to an instance profile assigned to it. Processes running on an EC2 instance with an IAM role associated with the instance profile can obtain temporary AWS API credentials by querying the EC2 IMS, at the well-known, link-local IP address 169.254.169.254. (Amazon Web Services, 2019).

EC2 instances are implicitly trusted to access the IMS at 169.254.169.254 and require no special HTTP request headers or authentication to do so. When EC2 instances are associated with an IAM role, two paths are available to discover the name of the role and to obtain an access key from temporary credentials with the privileges granted to that role, respectively:

1. iam/info
2. iam/security-credentials/*role-name*

Each of the detection research activities, therefore, looked for three observable payloads of interest: the request for the name of the IAM role attached to the vulnerable machine, the subsequent request to obtain a credential from the EC2 IMS, and the response payload from the EC2 IMS, which contains a temporary access key.

3.2. Detection using Amazon GuardDuty

AWS describes GuardDuty as “a continuous security monitoring service” (Amazon Web Services, 2019), although it is not a traditional intrusion detection or prevention tool. The service provides the ability for administrators to upload lists of trusted or malicious IP addresses which guide its assessments of findings, although it provides no other configuration options for its built-in finding types. While GuardDuty generates findings relatively quickly after events occur, it is not a real-time detection mechanism.

In the first portion of this test, a request was made to https://*site*/?url=http://169.254.169.254/latest/meta-data/iam/info which resulted in the following output of the EC2 IMS response:

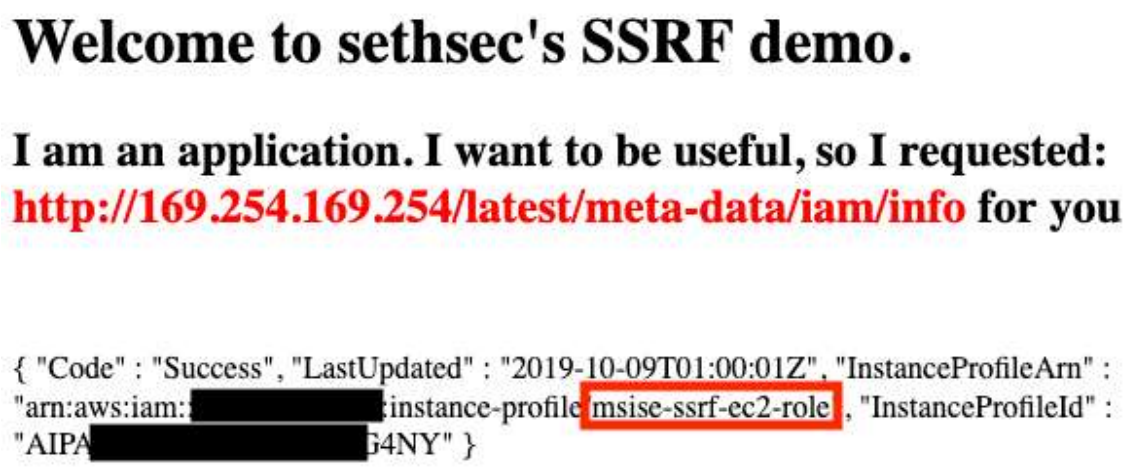

Figure 3 Vulnerable application exposing EC2 IAM Role after SSRF

For a period of up to one hour later, GuardDuty registered no reconnaissance findings for the request to gather the name of the role, which is the first step in retrieving a temporary credential from the IMS.

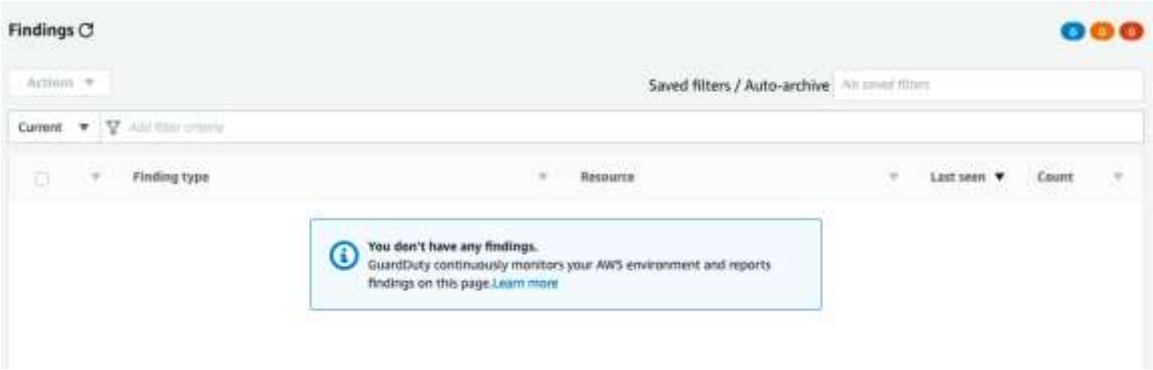

Figure 4 GuardDuty without findings after an SSRF attack

Subsequently, a call to https://*site*/?url=http://169.254.169.254/latest/meta-data/iam/security-credentials/msise-ssrf-ec2-role retrieves a temporary access credential as shown in Figure 5.

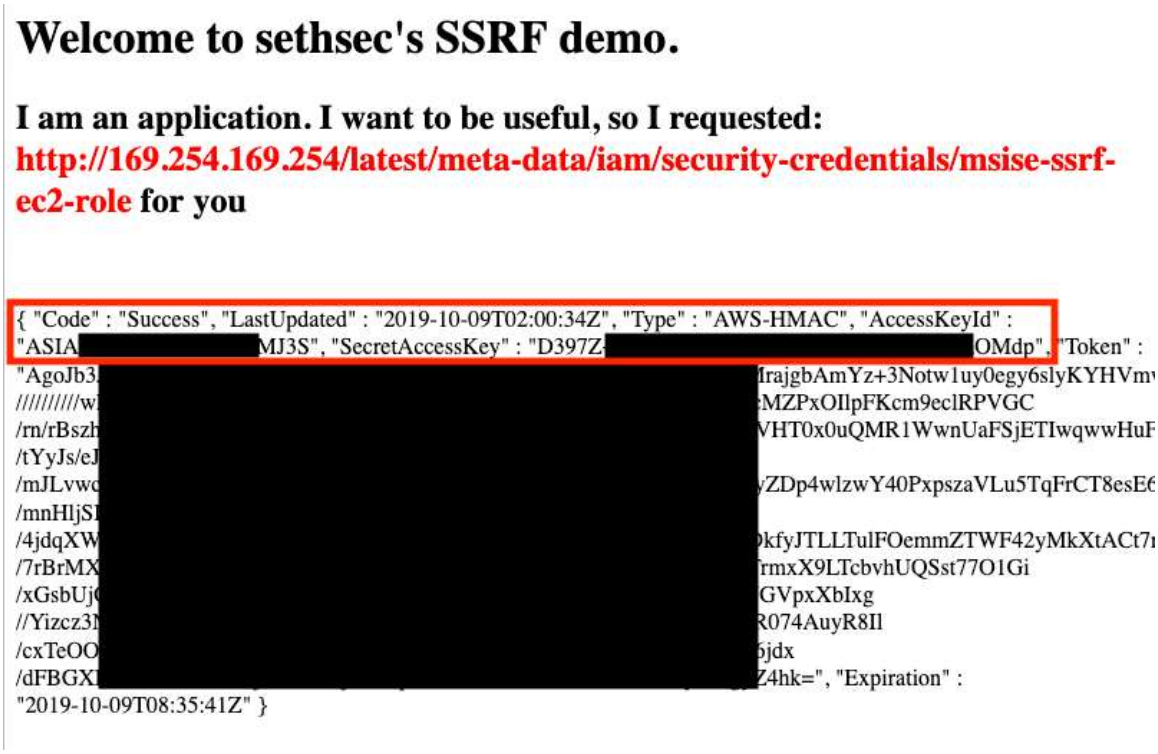

Figure 5 Exfiltrated AWS temporary credential after successful SSRF

Again, after an hour of observation, GuardDuty did not identify the SSRF attack. As a final post-exploitation activity, the following commands were run from a remote computer to use the stolen credential to list S3 buckets in the account:

```
$ export AWS_ACCESS_KEY_ID=ASIA....MJ3S
$ export AWS_SECRET_ACCESS_KEY=D397Z....OMdp
$ export AWS_SESSION_TOKEN=AgoJb3....Z4hk=
$ aws s3 ls
2019-09-15 17:10:49 msise-ssrf
```

Figure 6 Post-SSRF exploitation commands to confirm AWS API access

Six minutes after the use of the stolen credential, GuardDuty did identify the post-SSRF attack activity as shown in Figure 7.

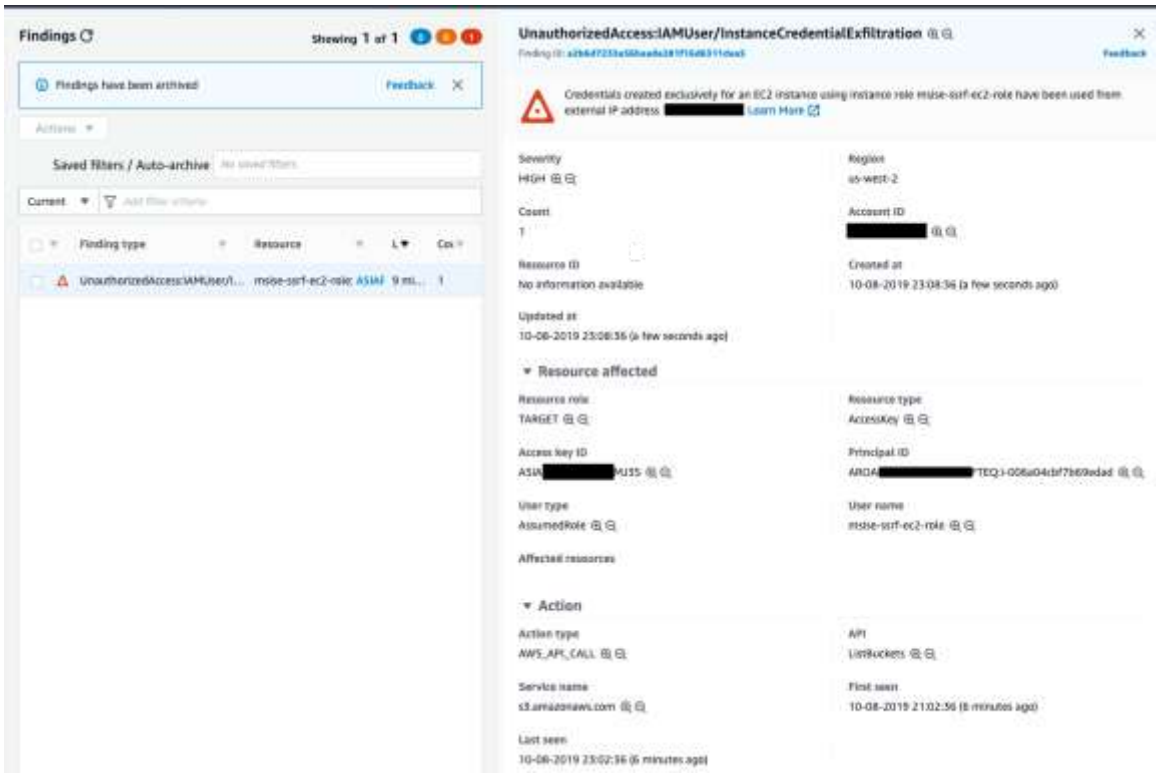

Figure 7 GuardDuty post-SSRF exploitation finding upon external credential use

Ultimately, GuardDuty failed to detect the SSRF attack and exfiltration of credentials. If one uses credentials outside of the EC2 instance that generated them, GuardDuty generated a finding of type UnauthorizedAccess:IAMUser/InstanceCredentialExfiltration. However, because this finding is not the SSRF attack itself, a threat actor who can achieve successful SSRF may already be privileged enough to perform other malicious actions. If the attacker can issue additional commands using the credential through command injection, GuardDuty may not identify the post-exploitation activity.

GuardDuty is ineffective because, according to its documentation, the sources it analyzes to generate findings do not include the payload content, but rather only metadata in the form of VPC Flow Logs, AWS CloudTrail event logs, and DNS logs. This metadata is not sufficient to detect an SSRF targeting a link-local address, which does not show up in VPC Flow Logs and does not initiate a DNS resolution request.

3.3. Detection using VPC Traffic Mirroring

VPC Traffic Mirroring provides a copy of network traffic, both inbound and outbound, on an Elastic Network Interface (ENI) attached to the desired source to the ENI of a designated target. The service provides only the mechanism to obtain network traffic but does not otherwise process or analyze it. It is incumbent on the implementer to deploy, configure, and maintain an IDS in an out-of-line configuration to generate events of interest.

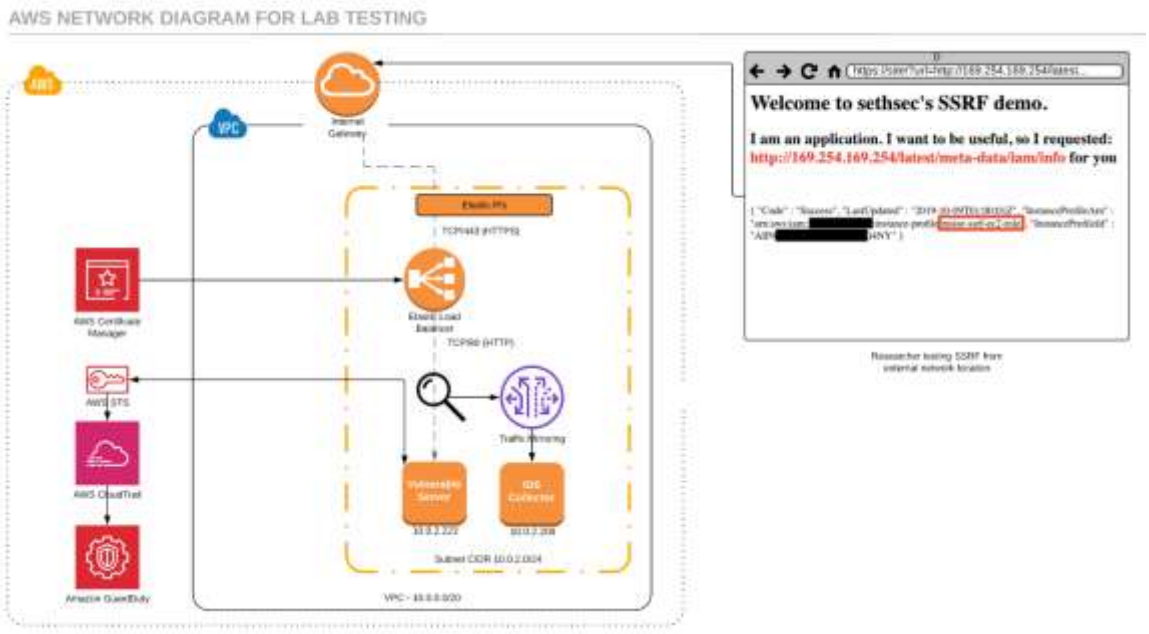

Figure 8 Network diagram of lab environment

VPC Traffic Mirroring is straightforward to set up, given both the source and target of a mirroring session are an AWS Nitro-based instance. While AWS VPC Traffic Mirroring documentation uses the terms “source” and “target”, the “target” running the IDS will be able to observe both requests to and responses from ENI attached to the “source”. The researcher created a traffic mirroring session that read all TCP traffic using a 0.0.0.0/0 source and destination in the Traffic Filter— from the “source” ENI on the instance with the vulnerable service to a target with a second ENI dedicated. AWS still evaluates the security group rules of the target, in addition to the Traffic Filter rules for payloads observed by the source.

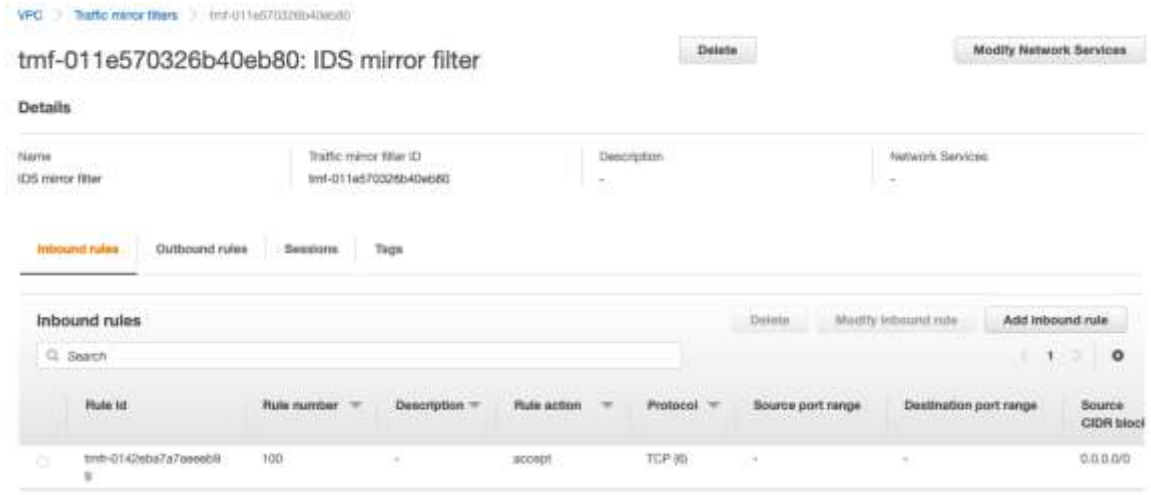

Figure 9 AWS VPC Traffic Mirroring filter configuration

The researcher verified the preliminary configuration by testing `sudo tcpdump -vi eth1`, where eth1 was a second ENI dedicated to the ingest interface for the mirror session. The resulting output demonstrated content payloads of an SSRF attack request and response, indicating that Snort, Zeek, and other open-source tools can detect them.

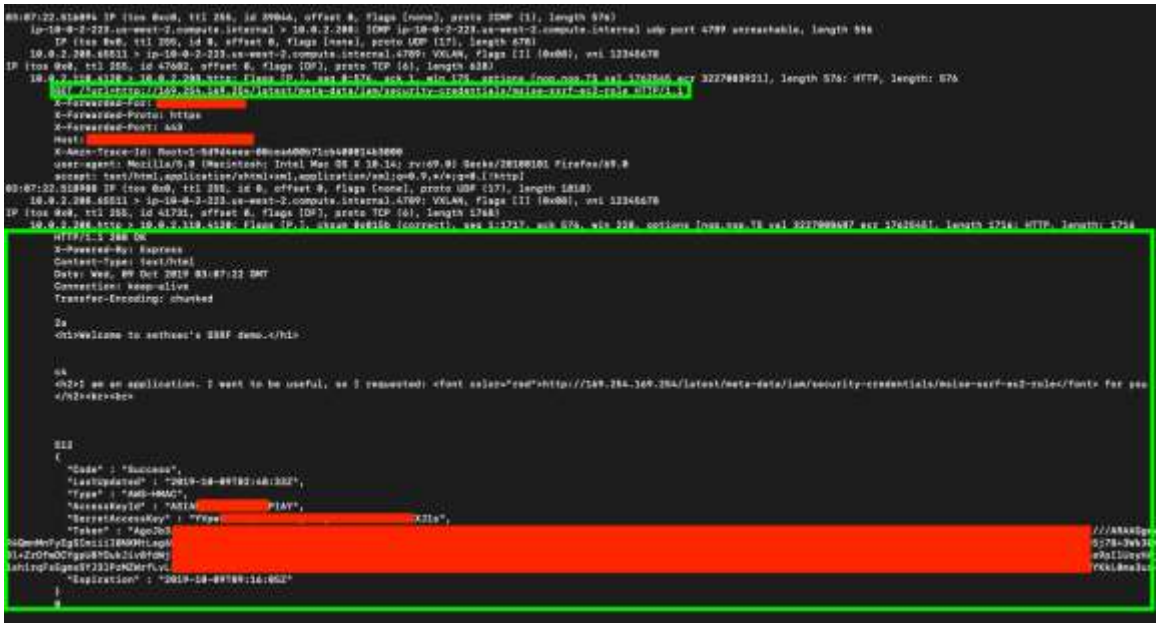

Figure 10 Redacted tpcdump output on target once VPC Traffic Mirroring enabled

Because VPC Traffic Mirroring encapsulates mirrored traffic in a VXLAN header, an IDS deployed to a traffic mirroring target must be able to parse the VXLAN conventions to inspect and alert on payloads. While Snort 2.9.14.1 can match content in

an IP packet, it cannot decapsulate VXLAN, preventing the use of the Stream preprocessor in more precise rules that account for a TCP flow. For tests performed with network intrusion detection tools for VPC Traffic Mirroring, the researcher confirmed the traffic mirroring feature did not capture local requests to EC2 IMS over the link-local 169.254.169.254 address.

3.3.1. Detecting SSRF using Zeek

Zeek, formerly named “Bro”, is a network traffic analysis tool with a flexible, event-based model and supports scripting capabilities. While Zeek is a specialized event-based correlation tool, version 3.0 supports both VXLAN decapsulation and signature-based detection using regular expressions. Given these capabilities, both an SSRF request and response can be identified independently with the following two Zeek signature definitions:

```
signature aws-ec2-ims-request {
    ip-proto == tcp
    dst-port == 80
    payload /.*meta-data\/iam\/security-credentials/
    event "EC2 Instance Metadata path in request, SSRF attempt"
}

signature aws-ec2-ims-response-access-key {
    ip-proto == tcp
    src-port == 80
    payload /.*\"SecretAccessKey\" :/
    event "Potential AWS IAM temporary credential in HTTP
response, successful SSRF exploitation"
}
```

Figure 11 Zeek signature to detect SSRF targeting AWS EC2 IMS (ssrf.sig)

The first signature detects both a request to enumerate a role name and the request to obtain a temporary access credential since both requests share the same partial path. Importantly, the “latest” portion of the path is omitted from the signature, as a specific IMS protocol version can also be targeted to obtain credentials. Running Zeek using the

command "`zeek -r request.pcap -s ssrf.sig`" generates a signatures.log file which details the detections:

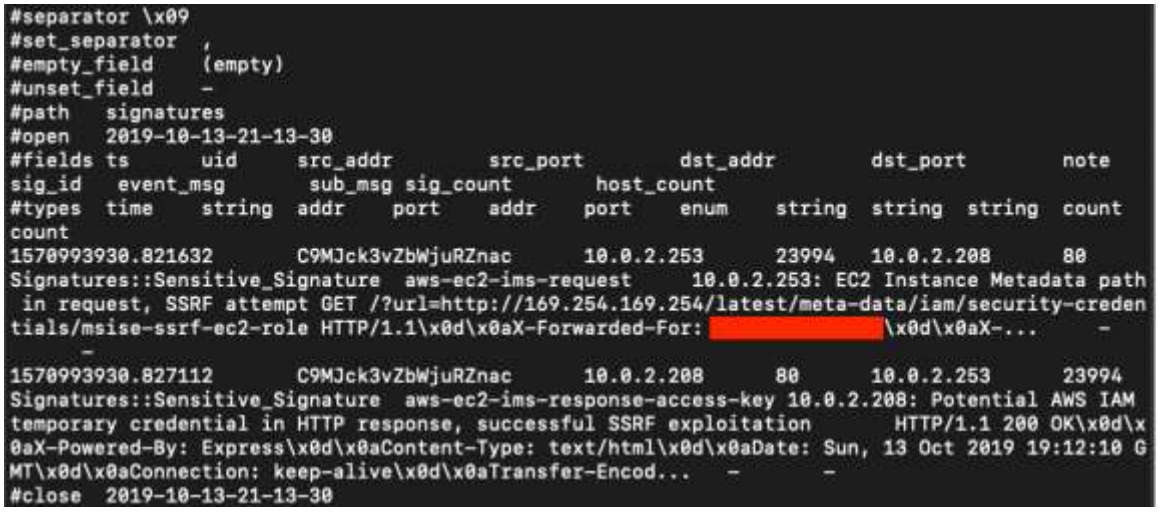

Figure 12 Zeek signatures.log output showing SSRF detection

3.3.2. Detecting SSRF using Suricata

While Snort currently does not support VXLAN decapsulation, Suricata 4.1.5 does. Suricata is an open-source intrusion detection and prevention system with many of the same signature-based capabilities of Snort. While the default signature set in Suricata does not detect attempted access of the EC2 IMS, by enabling the VXLAN decoder and defining signatures for the metadata endpoint, it could detect SSRF attempts.

```
alert ip any any -> $HOME_NET 80 (msg:"AWS EC2 IMS Recon";
sid:10000001; rev:001; flow:to_server; content:"/meta-
data/iam/security-credentials";)
alert ip $HOME_NET 80 -> any any (msg:"AWS EC2 IMS Credential
Exfil"; sid:10000003; rev:001; flow:to_client,established;
content:"\"SecretAccessKey\" :";)
```

Figure 13 Suricata rules that detect SSRF requests and responses with temporary credentials

Because Suricata can correctly interpret VXLAN encapsulation, HTTP activity does not appear as UDP traffic but rather as the underlying TCP streams. Running Suricata using the command "`suricata -r request.pcap`" generates a /var/log/suricata/fast.log file which details the detections:

Figure 14 Suricata alert logs demonstrating detection of SSRF attempts on the EC2 IMS and credential exfiltration

3.4. Detection using iptables

`iptables` is a packet filter that is available on Amazon Linux. This rule-based facility allows an administrator to define parameters for matching a packet and then specify an action for what to do when a match occurs. Available actions include logging, forwarding, and dropping the packet. While observing the `connect` syscall is a rather unspecific way to monitor network activity, `iptables` provides matching conditions that include source and destination IP addresses, IP protocol, and destination port. Furthermore, it can match the UID, GID, PID, and SID of the packet creator.

Given this capability, where an administrator has control over the EC2 instance where unique UID's are applied separately to processes, it is possible to reliably detect SSRF attacks with a low rate of false positives minimally by differentiating by PID, or in environments where processes are separated by individual purpose service principals, by UID or GID. Depending on the control environment, there are multiple methods to grant a process the ability to bind to a privileged port with a separate UID. To test whether `iptables` can answer the question as to whether it can be used to detect SSRF, the researcher granted a system capability to the Node.js binary to permit it to bind to ports below 1024 with the command `sudo setcap CAP_NET_BIND_SERVICE=+eip /usr/bin/node`. However, in production environments, alternative strategies such as chroot or containerization, or mapping a load balancer's inbound HTTP or HTTPS listener to a non-privileged port on the host through port address translation, which AWS Elastic Load Balancers natively support, may be more appropriate.

With a known UID (1001 in this case) that acts as a server to handle user requests which we do not expect to call the IMS, one can enter a detection rule using `iptables` with the following command: `sudo iptables -A OUTPUT -p tcp --dport 80 -d 169.254.169.254 -m owner --uid-owner 1001 -j LOG`

Once configured, `syslog` or `dmesg` logs all access to the IMS that originate from the Node.js server under test, as evidenced in Figure 15:

```
[428320.307558] IN= OUT=eth0 SRC=10.0.2.208 DST=169.254.169.254 LEN=60 TOS=0x00 PREC=0x00 TTL=255 ID=34334 DF PROTO=TCP SPT=42968
DPT=80 WINDOW=26883 RES=0x00 SYN URGP=0
[428320.314511] IN= OUT=eth0 SRC=10.0.2.208 DST=169.254.169.254 LEN=52 TOS=0x00 PREC=0x00 TTL=255 ID=34335 DF PROTO=TCP SPT=42968
DPT=80 WINDOW=211 RES=0x00 ACK URGP=0
[428320.322086] IN= OUT=eth0 SRC=10.0.2.208 DST=169.254.169.254 LEN=242 TOS=0x00 PREC=0x00 TTL=255 ID=34336 DF PROTO=TCP SPT=42968
 DPT=80 WINDOW=211 RES=0x00 ACK PSH URGP=0
[428320.329370] IN= OUT=eth0 SRC=10.0.2.208 DST=169.254.169.254 LEN=52 TOS=0x00 PREC=0x00 TTL=255 ID=34337 DF PROTO=TCP SPT=42968
DPT=80 WINDOW=234 RES=0x00 ACK URGP=0
[428320.347703] IN= OUT=eth0 SRC=10.0.2.208 DST=169.254.169.254 LEN=52 TOS=0x00 PREC=0x00 TTL=255 ID=34338 DF PROTO=TCP SPT=42968
DPT=80 WINDOW=234 RES=0x00 ACK FIN URGP=0
```

Figure 15 dmesg output after an iptables LOG rule implemented to log IMS accesses from a publicly accessible server process.

Moreover, this research identified a second `iptables` rule that, when appended to the `iptables` rule chain after the LOG rule, could also prevent SSRF by rejecting the outbound with the command: `sudo iptables -A OUTPUT -p tcp --dport 80 -d 169.254.169.254 -m owner --uid-owner 1001 -j REJECT`

With this "reject" rule applied, the vulnerable test program executed but logged an internal error when the local firewall rule rejected the SSRF connection attempt. Because the program properly handled the error conditions, the SSRF attempt rejection did not affect the end-user experience, and the Node.js process reported the following error to the console:

```
New request: /?url=http://169.254.169.254/latest/meta-data/iam/security-credentials/msise-ssrf-ec2-role
error
```

Figure 16 Application error message when iptables blocked SSRF access via a REJECT rule

With access to the shell of an EC2 instance, a system administrator can use principal segregation and `iptables` to both detect and prevent SSRF access to the IMS. Application programmers also have an opportunity to catch and log connection errors, which can provide an alternative method for security personnel to identify blocked SSRF attempts. However, 'black box' virtual appliances provided to administrators to deploy into their environments, such as from the AWS Marketplace, require their publishers to implement appropriate SSRF protections since customers cannot configure the `iptables` host-based control on them.

3.5. Detection using auditd

`auditd` is a package of system logging facilities that enables the capture of low-level audit activity. It can capture essential information about a wide variety of system calls (or "syscalls") and forms the basis of many security event detection systems that correlate and analyze such data for potential malicious activity.

There are limited opportunities to narrow the false positive rate of detection based on syscalls of interest to this attack. Given that operating systems provide a small number of frequently used syscalls, an HTTP request for a single SSRF attack is indistinguishable from legitimate activity. For one, an SSRF attack connects to a known host, but does so from within the process context of the vulnerable server component, usually a webserver. If a server process has the need to spawn additional processes, auditing of the `execve` syscall, which spawns a new program, would not be reliable SSRF indicator. An SSRF attack on the IMS targets a well-known destination, 169.254.169.254, and does not need to resolve a hostname to an IP address, rendering auditing of the `gethostname` syscall useless to detect SSRF. The `connect` Linux syscall can detect SSRF activity directed at the EC2 IMS link-local address of 169.254.169.254. By adding an audit rule similar to the following, `auditd` records socket connections in the audit logging directory:

```
auditctl -a always,exit -F arch=b64 -S connect
```

Figure 17 Command to add an auditd rule to log the connect() syscall

The audit.log file contains verbose information about the process context that performs any audited action, and connect syscalls contain UID, EUID, GID, and PID annotations, along with an encoded `saddr` argument which encodes the socket family, port number, and IP address in a hexadecimal format. For the EC2 IMS, this value is: `saddr=02000050A9FEA9FE0000000000000000`

Just as observed with `iptables`, security engineers can use audit records to observe connect requests targeting the IMS which originate from unexpected effective user identities. This activity would identify SSRF requests with a low false-positive rate provided administrators use distinct UIDs and GIDs for processes that need IMS access and for those that do not.

3.6. Discussion of other known detection techniques

3.6.1. Inline Firewall Services and Appliances

Intrusion detection and prevention systems (IDPS) within an AWS virtual private cloud (VPC) that can decrypt and analyze traffic have the potential to observe and block attempts to access the IMS. However, because cloud environments use software-defined

networks (SDNs) which do not support both broadcast network addresses and which traditionally do not support port mirroring or spanning to capture traffic at a network switch or router device, these strategies required inline appliances, host-based intrusion detection systems (HIDS), or host-based traffic forwarding to IDS collectors.

Inline IDS systems are problematic in cloud environments because of their propensity to become overloaded, inability to be effectively load-balanced given SDN constraints, and deployment cost. Engineers must deploy numerous collectors for comprehensive ingress and egress coverage (C. Mazzariello, 2010). Instead, a common practice has been to implement IDPSs or transparent proxies at lower-bandwidth egress points for explicit whitelisting and blocking functionality.

Host-based traffic forwarding is problematic in AWS as bandwidth limits are associated with instance type, not the network interfaces attached to them (Amazon Web Services, 2019). For this reason, bandwidth consumed to clone and forward traffic to an IDS collector counts against the available capacity of an instance to serve its intended business need. While others have detailed the use of Snort and Zeek using host-based traffic forwarding (Reese, 2018), cost-sensitive and highly-scalable cloud workloads often do not provide for additional application-handling instances solely to maintain network capacity for a forwarding-based IDS solution. Similarly, HIDS solutions suffer from resource limitations in cloud environments that can result in high compute utilization and packet loss (P. K. Shelke, 2012).

3.6.2. AWS WAF

AWS provides a web application firewall (WAF) offering, named AWS WAF, that operates similarly to inline appliance-based WAF solutions. Implementers must use the AWS content delivery network, CloudFront, to enable AWS WAF. Unlike specialized WAF appliances, the AWS WAF comes with comparatively few built-in rules to detect OWASP threats to applications, namely for SQL injection and Cross-Site Scripting signatures. However, WAF administrators can configure string and regular expression matching conditions that can detect and block SSRF, and other security engineers have demonstrated this previously (Sripati, 2019).

Generally, relying on highly distributed systems like CDNs to provide extension points for content inspection to detect SSRF activity can be expensive, as the barrier to entry may not be economical for all enterprises or all publicly exposed workloads (Modi, 2017). However, in AWS workloads already using the CloudFront CDN, using AWS WAF ACLs to match on the IMS link-local address of 169.254.169.254 can be a meaningful layered defensive control. Since the AWS WAF operates on the HTTP transport layer, it may be less subject to false positives for this address which may occur in binary data, such as image files.

3.6.3. Comparing EC2 IP addresses at AssumeRole with Credential Use

Since an EC2 instance assumes its associated IAM role when it starts, the internal AWS EC2 service call to `AssumeRole` is a logged event in AWS CloudTrail. Since the IMS generates unique temporary access keys for each instance at least every six hours, one could record the IP addresses of an EC2 instance when it assumes an IAM Role and correlate them with other API calls logged in CloudTrail. This technique provides for the observation of temporary credential usage outside the instance that generated them, which may indicate an SSRF attack was successful in providing unauthorized access (Bengtston, 2018).

However, an attacker who can access the IMS to obtain credentials can likely access the AWS API from within that same instance, which would evade this detection technique (Fernandez, 2019). This detection technique of recording access credential generations and correlating their usage over time would also be complex to implement and operate in environments that leverage multiple regions for resiliency since CloudTrail logs, as well as the AWS Security Token Service that creates temporary credentials, can operate both regionally and globally (Amazon Web Services, 2019). Furthermore, the trend towards managed containerized service offerings like AWS Elastic Kubernetes Service (EKS) that multiplex and rotate many containers and IP addresses on a single EC2 instance, each with unique IAM roles, introduce significant challenges when correlating probable SSRF activity with this method.

4. Recommendations and Implications

4.1. Recommendations for Practice

4.1.1. Configuration Best Practices

Administrators of Amazon Web Services should be familiar with and follow published best practices, including the AWS Well-Architected Framework documentation. Specifically, administrators should enable VPC Flow Logs and Amazon GuardDuty to detect post-SSRF IMS credential use attempts outside the environment. A variety of continuous monitoring tools exist to measure the conformance of AWS accounts to the AWS Well-Architected Framework and Center for Internet Security Amazon Web Services Foundations Benchmark, such as Cloud Conformity and Prowler, respectively. Security teams should deploy detective tools to monitor for the implementation of these controls across AWS accounts. Especially for environments that depend on manual administration instead of a high degree of automation, changes can introduce variance and the potential for security misconfiguration.

Unless necessary for an EC2 instance to access the AWS API, administrators must not attach an IAM role to an instance. When necessary, IAM policy authors should carefully construct statements using a least-privileges methodology when attaching them to roles used for this purpose. Authors should also utilize IAM policy conditions that limit service access, so temporary access credentials issued by the IMS and obtained through an SSRF attack have limited value when outside the environment. As with all technical controls, knowledgeable teams should review IAM role use and IAM policy statements regularly.

As a layered control, host-based controls like `iptables` that log and prevent access to the IMS except when originating from processes that have a known, legitimate need to read instance metadata or obtain temporary access credentials to act on behalf of an instance to access other AWS services. Additionally, given that VPC Traffic Mirroring now allows established intrusion detection tools to monitor and alert on suspicious traffic, security engineers should review the Zeek or Suricata rules developed as part of this research for inclusion in IDS deployments in Amazon Web Services, particularly for instances that serve external traffic. Of the detection mechanisms

evaluated in this research, only the host-based packet filter `iptables` and VPC Traffic Mirroring using Zeek or Suricata were found to detect SSRF at the point of credential theft from the IMS, before credentials were used to attempt unauthorized access or modification of cloud environments. While both were effective, VPC Traffic Mirroring is superior given its ability to provide detective coverage of all AWS EC2 resources, including virtual appliance to which AWS account holders have no interactive login or administrative access to configure directly.

4.1.2. Effectiveness of Layered Detection Techniques

Given the relatively low cost of enabling VPC Flow Logs and GuardDuty, and while this research observed detection capabilities that were limited to post-SSRF exploit activities, organizations with sensitive data on the AWS cloud should strongly consider enabling this service and monitoring it closely, at least for the UnauthorizedAccess:IAMUser/InstanceCredentialExfiltration finding type.

However, because post-SSRF exploit activity can happen much later in the timeline of a successful attack, monitoring activity from the host using `iptables` and `auditd` is a logical next step in expanding the detective capability to identify SSRF attack attempts. Environments with sensitive data are generally required to have centralized logging facilities by regulatory mandate or industry compliance standards, and by using automation to deploy `iptables` to log and alert attempts from servers without a need to access the IMS from a centralized logging repository, administrators gain additional, not duplicative, visibility into SSRF attacks earlier in the process for machines they can directly control. Using `iptables`, administrators can configure a preventative control in addition to benefiting from layered detection.

VPC Traffic Mirroring and a traditional IDS solution requires additional cost and expertise to capture, store, analyze, and act on findings. However, in a cloud environment, the resource investment is not the only factor, as the IDS must be able to receive unencrypted network packets or have the capability to decrypt them. Generally, cloud configuration best practices dictate encryption be enabled both for data at rest and data in transit whenever possible. Especially for environments that leverage the automated certificate issuance that cloud provider certificate authorities offer, this can be

problematic as private keys are generally not available when using fully managed provider CA solutions. To gain the benefits of VPC Traffic Mirroring, organizations must weigh the tradeoff of TLS termination on cloud load balancers and mirror the unencrypted traffic on network interfaces behind them. Some firms may have legal or compliance requirements that limit their ability to adopt this layer of detection. When faced with that tradeoff, they should consider compensating controls if they have 'black box' virtual appliances deployed in their environments that SSRF attacks may target.

4.1.3. Challenges Regarding the Evolution of the SSRF Techniques

Because SSRF requests targeting the AWS EC2 IMS are addressing a static IP address, detecting link-local IP addresses like 169.254.169.254 in HTTP requests or the path iam/security-credentials is straightforward to encode into an IDS or WAF signature. Filter bypass techniques that obfuscate URLs may use mixed decimal, octal, and hexadecimal representation of the same address as 169.0xfe.0251.254. Simple static text or regular expression rule matches will fail to identify such evasion techniques unless administrators use a more sophisticated detection engine to identify unusual payloads.

More recent, novel approaches to evading detection with SSRF attacks include using specific Unicode characters to use the text-based nature of HTTP to split lines using protocol control indicators in the request to achieve injection capable of delivering an SSRF payload (Kelly, 2018). Other approaches, such as Request Smuggling, use malformed HTTP headers or leverage specific parsing behaviors of webservers to inject requests from the attacker into the start of another request to the same device (C. Linhart, 2005). First described in a publication from 2005, the flexible, forgiving nature of HTTP continues to breathe new life into HTTP parsing vulnerabilities, most recently by leveraging differing behaviors of Transfer-Encoding and Content-Encoding in a variant dubbed "HTTP Desync" to present new vectors for delivering injections, including SSRF (Kettle, HTTP Desync Attacks: Request Smuggling Reborn, 2019). One technique to defend against HTTP Desync may be to use content delivery networks, such as Cloudflare, Amazon CloudFront, or Imperva Cloud WAF, as TLS termination points if they offer stricter parsing of HTTP than the origins they protect.

4.1.4. Recommendations for Cloud Providers

Frequently, SSRF attacks leverage only control over the host and path of the HTTP request to succeed, and often cannot inject or manipulate other HTTP headers. In the case of Google Cloud's Compute Engine, it has implemented a required header and value of "Metadata-Flavor: Google" to access management APIs to reduce the success of SSRF attacks that cannot set this header (Google, 2019). AWS has no such protection, but if AWS added a requirement to access the EC2 IMS that would be difficult for an SSRF attack to mimic, it might reduce the potential for the success of SSRF.

The EC2 IMS provides a predictable target to attack with a well-known, link-local address. Currently, AWS WAF does not by include default content inspection rules for requests containing the 169.254.169.254 address or for responses containing a temporary access key. In the case of AWS, and generally for cloud providers, including WAF rules that cover SSRF use cases by default would improve the general detective capabilities for their customers.

Furthermore, while the temporary access key begins with characters 'ASIA' by convention, such keys are not implicitly limited to a caller. A temporary key obtained by an instance through IMS can issue API calls from outside of AWS infrastructure, in part, because before the advent of AWS PrivateLink in November 2017, AWS API calls within cloud networks were routed externally, over the public internet. (Amazon Web Services, 2017). If cloud providers, including AWS, required caller authentication to access sensitive operations and limited callers' use of temporary access credentials to the internal, it could reduce the damage potential of an SSRF attack.

4.2. Implications for Future Research

While many companies are migrating workloads to the cloud, traditionally, this has been termed a "lift and shift" operation as IT professionals trade virtual machines in on-premises infrastructures or co-located data centers for virtual machines operating as EC2 instances, using familiar operating systems and providing interactive administrative access. However, companies are also developing new workloads directly on the cloud, leveraging novel cloud services such as AWS Elastic Kubernetes Services, AWS Fargate, and AWS Lambda, all of which provide a computing environment but abstract away

administrative access, reserving that control plan for the cloud provider itself. Each of these offerings provides the opportunity for server-side request forgery against vulnerable applications or code, and each limits the ability of security professionals to use traditional tools to detect and prevent such attacks. Additional detection techniques may exist that are specific to cloud infrastructures, and security researchers should continue to search for layered controls that protect sensitive data from SSRF on cloud environments.

5. Conclusion

In summary, the cloud is not immune to server-side request forgery, and practical techniques exist to both detect and prevent attacks on vulnerable applications. Organizations that transmit or store sensitive data in the cloud should implement appropriate detective controls to identify SSRF exploitation attempts long before they learn about data exfiltration from a third-party. By leveraging these well-established tools on the cloud, security professionals can secure cloud workloads from SSRF targeting the AWS API and EC2 IMS, and the cloud can be an exciting, cost-effective, and safe place to deliver innovations at a massive scale and speed.

References

Amazon Web Services. (2017, September 7). *Announcing Network Load Balancer for Elastic Load Balancing*. Retrieved from About AWS: https://aws.amazon.com/about-aws/whats-new/2017/09/announcing-network-load-balancer-for-elastic-load-balancing/

Amazon Web Services. (2017, November). *AWS re:Invent 2017: NEW LAUNCH! Amazon EC2 Bare Metal Instances (CMP330).* Retrieved October 7, 2019, from Amazon Web Services channel on YouTube.com: https://www.youtube.com/watch?v=o9_4uGvbvnk

Amazon Web Services. (2017, November 8). *Introducing AWS PrivateLink for AWS Services*. Retrieved from About AWS: https://aws.amazon.com/about-aws/whats-new/2017/11/introducing-aws-privatelink-for-aws-services/

Amazon Web Services. (2019, September 6). *CloudTrail Concepts*. Retrieved from AWS CloudTrail User Guide: https://docs.aws.amazon.com/en_pv/awscloudtrail/latest/userguide/cloudtrail-concepts.html

Amazon Web Services. (2019, October). *Elastic Network Interfaces*. Retrieved from Amazon Elastic Compute Cloud User Guide for Linux Instances: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-eni.html

Amazon Web Services. (2019, September 21). *Instance Metadata and User Data*. Retrieved from AWS Documentation for EC2: https://docs.aws.amazon.com/en_pv/AWSEC2/latest/UserGuide/ec2-instance-metadata.html

Amazon Web Services. (2019, October 1). *What Is Amazon GuardDuty?* Retrieved from Amazon Guard Duty User Guide: https://docs.aws.amazon.com/guardduty/latest/ug/what-is-guardduty.html

Art, S. (2016, September 26). *Nodejs-SSRF-App*. Retrieved October 8, 2019, from GitHub.com: https://github.com/sethsec/Nodejs-SSRF-App/

Bengtston, W. (2018, August 8). *Netflix Cloud Security: Detecting Credential Compromise in AWS*. Retrieved from The Netflix Tech Blog: https://medium.com/netflix-techblog/netflix-cloud-security-detecting-credential-compromise-in-aws-9493d6fd373a

C. Linhart, A. K. (2005). *HTTP Request Smuggling.* Watchfire.

C. Mazzariello, R. B. (2010). Integrating a Network IDS into an Open Source Cloud Computing Environment. *2010 Sixth International Conference on Information Assurance and Security (IAS).* IEEE.

ERPScan. (2013, March 27). *SSRF DoS Relaying*. Retrieved from ERPScan.io Blog: https://erpscan.io/press-center/blog/ssrf-dos-relaying/

Fernandez, G. (2019, September 3). *Metadata abuse in AWS*. Retrieved from Technology with a business perspective.: https://medium.com/@gonfva/metadata-abuse-in-aws-d264274f5764

Google. (2019, October 01). *Storing and retrieving instance metadata*. Retrieved from Compute Engine Documentation: https://cloud.google.com/compute/docs/storing-retrieving-metadata

Institute of Information Security. (2015, April 16). *Server Side Request Forgery (SSRF)*. Retrieved from Institute of Information Security Blog: https://iisecurity.in/blog/server-side-request-forgery-ssrf/

Kelly, R. (2018, September 10). *Security Bugs in Practice: SSRF via Request Splitting*. Retrieved from Personal blog: https://www.rfk.id.au/blog/entry/security-bugs-ssrf-via-request-splitting/

Kettle, J. (2017, July 27). *Cracking the lens: targeting HTTP's hidden attack-surface*. Retrieved from PortSwigger Research: https://portswigger.net/research/cracking-the-lens-targeting-https-hidden-attack-surface

Kettle, J. (2019). *HTTP Desync Attacks: Request Smuggling Reborn.* PortSwigger Web Security.

Modi, C. &. (2017, March). Virtualization layer security challenges and intrusion detection/prevention systems in cloud computing: a comprehensive review. *Journal of Supercomputing, 73*(3), 1192-1234.

OWASP. (2017). *Top 10-2017 A1-Injection*. Retrieved from The Open Web Application Security Project: https://www.owasp.org/index.php/Top_10-2017_A1-Injection

P. K. Shelke, S. S. (2012, May). Intrusion Detection System for Cloud Computing. *International Journal of Scientific & Technology Research*(4).

rain.forest.puppy. (1998, December 25). NT Web Technology Vulnerabilities. *Phrack Magazine, 8*(54). Retrieved from http://phrack.org/issues/54/8.html

Reese, S. (2018, January 15). *Network Traffic Capture in Virtual Enviroments*. Retrieved from rsreese.com: https://www.rsreese.com/network-traffic-capture-in-virtual-enviroments/

Sripati, P. (2019, September 26). *How To Secure Web Applications With AWS WAF?* Retrieved from AWS Architect Certification Training: https://www.edureka.co/blog/secure-web-applications-with-aws-waf/