Playing hide-n-seek with AWS GuardDuty: Post-DNS era covert channel for C&C and data exfiltration

Sze Siong Teo
Independent Security Researcher

Disclaimer: The contents expressed in this presentation are solely my own and do not express the views or opinions of my employer.
Who am I?

- Developer
- Sysadmin
- Infosec guy

LinkedIn - [https://www.linkedin.com/in/ssteo/](https://www.linkedin.com/in/ssteo/)
Agenda

- DNS tunneling, NIDS, HIDS and SIEM
- Covert channel using SaaS or CDN
- Amazon GuardDuty evasion
- Proof-of-Concept and demo
- Common fallacy of AWS & mitigation tips
DNS Tunneling

- Direct TCP & IRC – Perl & C code, NetBus, BackOrifice, Sub7 (1990s)
- **DNS Tunneling - Bugtraq by Oskar Pearson (April 1998)**
- Attacker’s web server – Malware/backdoor programs (2000s)
- Reddit posts – iWorm botnet (2014)
- GitHub commits - Black Hat Python by Justin Seitz (2015)
- Twitter tweets and GitHub – Hammertoss by APT29 (2015)
- Instagram comments – Trojan horse by Turla Group (2017)
DNS Tunneling

- DNS traffic cannot be blocked easily
DNS Tunneling

• But... DNS queries are not encrypted

• What about DNS over HTTPS (DoH) and DNS over TLS (DoT)?
Network-based IDS (NIDS)

• Network-based IDS (E.g. Snort, Zeek, Suricata, etc.)
  • Able to detect DNS tunneling, but not 100%
  • Inspect packet header and unencrypted packet data (DPI)
  • Limited insight on encrypted channels like HTTPS
  • DPI does not scale well for high throughput networks
Host-based IDS (HIDS)

- Host-based IDS (E.g. OSSEC, Wazuh, ThreatStack, etc.)
  - Monitor filesystem integrity, processes, network and analyzes logs
  - Performance & stability concerns (Some use kernel-mode hooks)
  - Not very platform/distro agnostic (due to kernel module)
  - Can be noisy, might end up as a crying wolf
Security info and event management (SIEM)

- SIEM (E.g. ELK stack, Datadog, Splunk, AlienVault, etc.)
  - Ingest logs from various sources
  - Aggregate logs to gain traffic insights
  - Alerts and reactive actions can be triggered
  - Event search and investigation capability
Covert channel using SaaS or CDN

- Examples of popular enterprise SaaS
  - Source control – GitHub, GitLab, Bitbucket, etc.
  - APM tools - NewRelic, Dynatrace, AppDynamics, etc.
  - Monitoring tools – Datadog, Grafana, LogicMonitor, etc.

- Examples of popular CDN/WAF services
  - CloudFlare, Fastly, Imperva, etc.
Covert channel using SaaS or CDN

- Assume that the target server for data exfiltration
  - No ingress traffic from Internet, private network, no public IP
  - Only egress traffic to the Internet is through a NAT gateway
  - Uses Internet to fetch GitHub repos and get OS updates
  - Compromise it through supply chain attack (malware)
Covert channel using SaaS or CDN

• Setup for C&C/data exfiltration channel
  1. Gather the IP range of top 10 (or more) cloud services that
     • are widely used by many organizations
     • have publicly documented API
     • have ability to store and retrieve data
  2. Gather the IP range of top 5 CDN services
  3. Create SaaS accounts for API keys and setup C&C server behind CDNs
  4. Embed API keys and IP range data into the malware
Covert channel using SaaS or CDN

- Malware monitors OS connection table for at least 24 hours
  - Look for remote IP that matches any embedded IP range data
    - Linux - `/proc/net/tcp (IPv4), /proc/net/tcp6 (IPv6)`
    - Windows – `GetTcpTable (IPv4), GetTcp6Table (IPv6)`
  - Or... find repo host with IP that matches any embedded IP range data
    - `/etc/apt/sources.list.d/*`
    - `/etc/yum.repos.d/*`
Covert channel using SaaS or CDN

1. Found a matching SaaS IP? **Use it to blend in the C&C traffic**
   - Malware ← store/retrieve data → SaaS’s storage as data exchange medium
   - C&C Server ← store/retrieve data → SaaS’s storage as data exchange medium

2. Found a matching CDN IP? **Use it to blend in the C&C traffic**
   - Malware → request → CDN as proxy for C&C → request → C&C Server
   - Malware ← response ← CDN as proxy for C&C ← response ← C&C Server

3. No matching IP? **Pick a CDN, still less suspicious than directly to C&C Server**
   - Malware → request → CDN as proxy for C&C → request → C&C Server
   - Malware ← response ← CDN as proxy for C&C ← response ← C&C Server
Amazon GuardDuty evasion

- Amazon GuardDuty data sources
  - **CloudTrail** – logs all AWS API call and S3 data events
  - **VPC Flow Logs** - logs VPC traffic’s packet header without content
  - **Threat intel feed** – Known malicious IP addresses, etc.
  - **DNS logs** - logs all DNS requests to AWS DNS resolver

Note: IAM is the core service of AWS, all AWS API calls use it
Amazon GuardDuty evasion

- Evasion criteria, the C&C traffic **must not**
  - Call AWS API with compromised host’s IAM access (CloudTrail)
  - Communicate with any unknown IP address (VPC Flow Logs)
  - Communicate with any malicious IP (Threat intel feed)
  - Use DNS tunneling (VPC Flow Logs and DNS Logs)
Amazon GuardDuty evasion

- How the ideal "solution" looks like?
  - A medium for data exchange without leaving trace in CloudTrail
  - The medium for data exchange must be trusted by AWS
  - Simple to implement in malware without additional dependency
  - Uses standard HTTPS traffic for communication
Amazon GuardDuty evasion

Option #1 - Embed the IAM user access key of the attacker
• Can use any AWS resource of the attacker (E.g. S3, SQS, DynamoDB)
• \texttt{STS::GetCallerIdentity()} reveals AWS account ID and IAM username
• Need HMAC/SHA256 library to sign AWS API call at the compromised host

Note: This method only generates CloudTrail logs in the attacker’s AWS account
Amazon GuardDuty evasion

Option #2 – Setup C&C server behind AWS CloudFront (CDN)
- No API call signing is required (i.e. Embed API key and URL of C&C in malware) ✅
- EC2 initiated outbound connection to CloudFront may appear suspicious ❌

Note: Amazon Linux 2’s package repository uses S3 without CloudFront (i.e. https://amazonlinux-2-repos-[region].s3.[region].amazonaws.com)
Amazon GuardDuty evasion

Option #3 – Use attacker’s S3 bucket via pre-signed URL

• No API call signing is required, URL already has signature
• S3 is preferred over CloudFront due to AL2’s package repository using it
• Limited validity of 7 days, but workaround is possible
Proof-of-Concept and demo

1. Generate pre-signed URL for file upload every 24 hours and save into publicly accessible files with prefix of "cnc-channels-{number}".

2. Randomly pick one of the "channels" and send a session ID (random UUID) to request for a private session creation.

3. Periodically poll the bucket for any session creation request and create session based on received ID. Session is created using pre-signed URL for file upload with prefix of "sessions/{session-id}".

4. Wait until sessions/{session-id} is found and periodically poll for message from server stored as "server.msg/{session-id}"

5. Process message from server as command and store output as "client.msg/{session-id}" using pre-signed URL in #3

6. Process response message from the client at "client.msg/{session-id}" and display as command output

PoC code at https://github.com/sssteo/hitbseccconf2021ams-poc
Common fallacy of AWS & mitigation tips

1. Allowing all egress traffic to AWS services is safe
   - Use PrivateLink endpoint policy to restrict traffic to a specific S3 bucket

2. Using iptables in EC2 is the same as using security groups
   - It is recommended to use security groups over iptables

3. GuardDuty is good enough for overall security monitoring
   - Continuous tweaking of SIEM is the key to improve security visibility
Thank You

LinkedIn - https://www.linkedin.com/in/sssteo/