Docker for AWS Configuration Review

Docker

April 11, 2017 – Version 1.0

Prepared for

Riyaz Faizullabhoy

Prepared by

Raviv Cohen

While precautions have been taken in the preparation of this document, NCC Group the publisher, and the author(s) assume no responsibility for errors, omissions, or for damages resulting from the use of the information contained herein. Use of NCC Group's services does not guarantee the security of a system, or that computer intrusions will not occur.

Executive Summary

Synopsis

In February of 2017, Docker engaged NCC Group to conduct a security assessment of the Amazon Web Services (AWS) environment and infrastructure generated and configured by Docker for AWS. Docker for AWS provides an automatically configured installation of Docker swarm mode on Amazon's cloud services platform. This assessment was open-ended but was time-boxed at ten person-days of effort. Source code was not provided and the scope included only the configuration assembled by Docker for AWS.

Scope

NCC Group's evaluation focused on Docker for AWS, which is intended to allow Docker users to deploy Docker swarm mode on AWS both efficiently and securely. The combination of pre-built virtual machines and configurations are intended to provide a securely configured cluster with minimal effort. The scope of this assessment included the configuration for the Docker for AWS CloudFormation template, as well as basic network and host hardening on the deployed virtual machines.

Features of Docker and Docker swarm mode, such as general container security and Docker-internal network handling, were out of scope, thus this report should not be construed as an assessment of those features.

NCC Group performed testing on the most recent public version (Docker for AWS 1.13.1 (ga-2)), documented at https://docs.docker.com/docker-for-aws/.

NCC Group's assessment of Docker for AWS included evaluating the access control (Access Controls Review on page 4) and network configuration (Network Configuration Review on page 5) provided by Docker in relation to the capabilities offered by AWS and the virtual machines provisioned.

Key Findings

The assessment identified a medium risk access control issue and a couple of low severity network exposure issues:

- The "Docker-Proxy" IAM role, which is created by Docker, is too permissive. It allows nodes, and potentially containers, to modify inbound and outbound firewall rules via assigning load-balancer listeners.
- The AWS instance metadata service is accessible to Docker containers, exposing host-sensitive data.
- The Docker swarm mode communication ports are exposed to Docker containers, enabling them to perform dangerous management operations.

Limitations

As noted under the scope section, the assessment did not include testing of Docker swarm mode functionality. Additionally, this assessment covered only the standalone installation stable channel and did not cover any future integration with Docker Cloud or Docker Datacenter. Finally, no source code or build scripts were available for the virtual machine images used by Docker for AWS, restricting the assessment to the output/results and precluding review of the inputs or generation process.

Strategic Recommendations

Networking: Consider segregating the bridges used by Docker from the rest of the internal network space to avoid attacks on internal resources from containers.

Access Controls and Auditing: Docker for AWS currently lacks sufficient forms of access controls and auditing, as there is only a single SSH key provisioned with no built-in mechanisms to add or revoke users, limit user access, or audit user actions. Therefore, NCC Group recommends that enterprise users consider other Docker products, such as Docker Datacenter, for their production deployments.

CloudFormation: Consider updating the current recommended role and associated permissions for the Docker for AWS CloudFormation template. All resources should be properly constrained to the Docker for AWS Virtual Private Cloud and condensed so that only the minimal subset of permissions needed to execute the template are used.

Docker for AWS: Create and advertise a security related mailing list to update Docker for AWS users when new security features are available, as users will be responsible for updating installations on their own.

Dashboard

Target Metadata

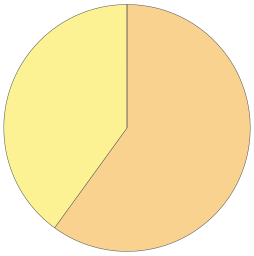
Name	Docker for AWS
Type	AWS Deployment Template - Docker for AWS 1.13.1 (ga-2)
Platforms	AWS
Environment	AWS VPC

Engagement Data

Type	Configuration Review
Method	Black-box
Dates	2017-02-06 to 2017-02-17
Consultants	1
Level of effort	10 person-days

Finding Breakdown

Critical Risk issues	0
High Risk issues	0
Medium Risk issues	3
Low Risk issues	2
Informational issues	0
Total issues	5



Category Breakdown

Access Controls	4	🟧🟧🟨🟨
Configuration	1	🟧

Key

Critical 🟥 High 🟥 Medium 🟧 Low 🟨 Informational ⬜

AWS Access Controls

AWS provides its customers with a fine-grained method of controlling access to its various services and resources through its AWS identity and access management (IAM) feature. IAM gives administrators the ability to control individual user and group access to all AWS services. Docker for AWS leverages the security model provided by Virtual Private Clouds (VPCs) by either creating its own VPC for its resources or by allowing users to deploy it into an existing VPC. Using the IAM permission model, administrators can restrict user access to sensitive AWS services (such as autoscaling) and instances created by Docker for AWS.

AWS allows users to create IAM roles. IAM roles define which IAM permissions an associated AWS service has. Docker leverages IAM roles by associating all worker and manager node EC2 instances created by Docker for AWS with the `Docker-ProxyRole-*`, where "*" is the Docker VPC ID. The `Docker-ProxyRole-*` role grants the ability to describe[1] any EC2 instance in the AWS account and assign load balancer listeners to any resource in the AWS account. NCC Group has determined that the `Docker-ProxyRole-*` role is too permissive. Docker for AWS should not assign this role to worker nodes as it could enable compromised containers to access the host instance's temporary credentials and issue AWS API calls; this is described further in finding NCC-DOCK_AWS_2017-002 on page 11.

Docker for AWS leverages an AWS CloudFormation template to manage its AWS deployment. It allows users to dynamically scale up and down their Docker swarm mode and perform rolling updates of the Docker swarm mode to the latest Docker image. However, NCC Group has identified some caveats in using the Docker-recommended server role described in the Docker for AWS deployment guide.[2] Information about the security implications can be found in finding NCC-DOCK_AWS_2017-001 on page 9. Users should consider employing the best practices recommendations outlined in Docker for AWS Deployment Best Practices on page 7.

Docker Access Controls

In the existing Docker for AWS implementation, Docker swarm mode management is supported via SSH and secured with a single public key provided during setup. As the user's expected source IP addresses and ranges are unknown, the SSH port is exposed by default to the entire Internet. Once signed in as the `docker` user, users are free to use sudo to perform actions as the `root` user on the Docker management host. This privilege elevation does not create significant additional attack surface; access to the Docker daemon is effectively equivalent to `root` access due to Docker's capabilities.

There are potential issues when groups of operators use the current revision of Docker for AWS to deploy swarms of containers; since all administrators log in as the `docker` user, the existing configuration makes it difficult to track which user initiated an action. Additionally, the default configuration requires all administrative users to use the same SSH key, making it difficult to rotate and revoke keys. Organizations wishing to have multiple people administer a single Docker for AWS cluster should strongly consider adding separate SSH keys for each of those administrators to the manager node's `authorized_keys` file, rather than sharing one key. It should be noted that this does not provide accountability for individual users, but instead makes removing access much simpler.

Docker for AWS does not offer, in its current model, any strong form of access controls or auditing and logging mechanisms to Docker nodes, due to the use of a single SSH key outlined above. Docker for AWS should be used in environments where such security requirements are not required. Enterprise users should consider Docker Datacenter for AWS[3] and Docker Cloud in AWS[4] as they potentially offer more fine-grained access control then Docker for AWS.

[1] https://docs.aws.amazon.com/cli/latest/reference/ec2/describe-instances.html
[2] https://docs.docker.com/docker-for-aws/iam-permissions/
[3] https://aws.amazon.com/docker/docker-datacenter/
[4] https://docs.docker.com/docker-cloud/infrastructure/cloud-on-aws-faq/

AWS publishes a set of security best practices[5] and supplies network access controls that provide users with the ability to forgo the traditional public IP address with firewall configuration. AWS allows users to create Virtual Private Clouds (VPCs) that provide a logically isolated network to deploy the user's resources. The VPC and the resources deployed to it are logically isolated using security groups that prevent any external network access to that resource by default.

Docker for AWS leverages and follows the set of outlined best practices: a VPC is created, and all virtual machines are deployed on the isolated VPC and exposed only via explicitly configured load balancers. This ensures that only expected ports are exposed to the Internet and it additionally uses AWS's controls to prevent further access.

Created Security Groups

Docker for AWS creates the following security groups and rules, where "*" is the Docker VPC ID:

1. `Docker-ManagerVpcSG-*`: Port 22 is open to the world, and ports 2377, 7946, and 4789 are open to Docker worker nodes. This group has no outbound restrictions and is applied to Docker manager nodes.

2. `Docker-SwarmWideSG-*`: All ports are open to the VPC. It has no outbound restrictions and is applied to Docker manager nodes.

3. `Docker-NodeVpcSG-*`: All ports are open to the VPC. It blocks outbound traffic to port 2375 and is applied to Docker worker nodes.

4. `Docker-ExternalLoadBalancerSG-*`: No ports are whitelisted and no listener is created by default. It allows for all internal resources to communicate with load balancers and is applied to Docker-created AWS elastic load balancers.

Public Network Configuration

Docker for AWS makes use of the default VPC security group created by AWS when a new VPC is deployed.[6] The default security group allows only inbound traffic from VPC-assigned resources. Docker for AWS then attaches an Internet Gateway (IG) to the created VPC that connects it to the Internet and enables public access. External access to each of the nodes within the VPC is then governed by the security group protections outlined above. Since no listener is assigned to the load balancer by default, the only default access into the VPC is to port 22 on the manager nodes.

These nodes expose public-key SSH (OpenSSH) for remote administration and have SSH password authentication disabled. While directly exposing SSH to the Internet is a concern in many environments (primarily when password authentication is possible), this configuration provides security comparable to many VPNs as only a private key will grant access and OpenSSH has had no significant publicly known pre-authentication vulnerabilities for an extended period of time.

Intra-Swarm Network Configuration

Docker swarm mode generally expects that all hosts are able to communicate with one another relatively freely.[7] While AWS allows users to firewall individual hosts or groups of hosts within a VPC through the use of security groups, Docker for AWS does not currently attempt to limit swarm-wide access, and the `Docker-SwarmWideSG-*` security group exposes manager nodes to the swarm. To prevent access to the internal Docker daemon listener, Docker for AWS blocks worker nodes from making outgoing connections to port 2375.

Container Network Configuration

Docker swarm mode does not constrain container network access. The bridged network[8] containers are configured with allows any container running in the Docker for AWS environment the ability to access ports used by Docker swarm mode for coordinating and monitoring the Docker swarm mode hosts. NCC Group has filed this issue as finding NCC-DOCK_AWS_2017-003 on page 13. This is not expected to cause significant security issues, as all accessible ports

[5] https://d0.awsstatic.com/whitepapers/Security/AWS_Security_Best_Practices.pdf
[6] https://docs.aws.amazon.com/AmazonVPC/latest/UserGuide/VPC_SecurityGroups.html#DefaultSecurityGroup
[7] https://docs.docker.com/engine/userguide/networking/overlay-security-model/
[8] https://docs.docker.com/engine/userguide/networking/#default-networks

reportedly require authentication with secrets that would not be known to containers.[9] Nonetheless, security best practices state that the containers should not need to be able to contact the Docker swarm mode hosts.

AWS provides an instance metadata service at the link-local URL http://169.254.169.254/ to expose configuration and AWS-specific data to instances. AWS makes heavy use of the instance metadata service, which can supply sensitive information, such as temporary credentials if an instance is associated with an IAM role. This service is exposed to all containers providing, at a minimum, information about the host they are running on. As Docker for AWS does associate its instances with a role, the current configuration presents an undue security risk. More information regarding this issue see finding NCC-DOCK_AWS_2017-004 on page 14 and finding NCC-DOCK_AWS_2017-002 on page 11.

Note: Overlay networking used by Docker swarm mode (such as VXLAN) was outside the scope of this assessment and is not considered part of the Docker for AWS infrastructure.

[9] This authentication is part of Docker swarm mode and thus out of scope for this assessment.

Docker for AWS follows many industry-standard and AWS[10] best practices for securing the AWS resources it creates; however, users of Docker for AWS can increase the security of deployed resources in some areas. In particular, NCC Group recommends:

Before setting up Docker for AWS

Do not create the recommended CloudFormation Role

Docker for AWS recommends that the user create and assign a role with a very broad set of permissions to CloudFormation.[11] NCC Group has cited in finding NCC-DOCK_AWS_2017-001 on page 9 that the current role is too permissive. Instead users should execute the CloudFormation template without assigning a CloudFormation IAM role. This will result in the CloudFormation template executing within the context of the executing user's IAM user permissions. This prevents users with limited permissions from using CloudFormation to execute actions that the user does not have permission to perform.

After setting up Docker for AWS

Restrict SSH access

If possible, restrict access to the master SSH port by IP address or range in the AWS portal. For example, this can be implemented by limiting connections to a corporate VPN source IP address. Even though the configuration only enables SSH public key authentication, limiting the attack surface here can protect against pre-authentication vulnerabilities in OpenSSH.

Restrict container access to the AWS metadata service

Restrict access from the containers running on Docker for AWS workers to the AWS metadata API. More information is available in finding NCC-DOCK_AWS_2017-004 on page 14.

Do not share an SSH key

If multiple users will be administering this Docker for AWS installation, consider adding each user's SSH key to the `docker` user's `authorized_keys` file for the Docker manager host, rather than forcing all users to share the same SSH key. Alternatively, look into options such as Docker Cloud or Docker Datacenter (when available) for managing access control to your Docker for AWS cluster.

[10] https://d0.awsstatic.com/whitepapers/Security/AWS_Security_Best_Practices.pdf
[11] https://docs.docker.com/docker-for-aws/iam-permissions/

Table of Findings

For each finding, NCC Group uses a composite risk score that takes into account the severity of the risk, application's exposure and user population, technical difficulty of exploitation, and other factors. For an explanation of NCC Group's risk rating and finding categorization, see Appendix A on page 15.

Title	ID	Risk
Privileged IAM Role Passed to CloudFormation Stack	001	Medium
Docker-Proxy Role Passed to Worker Nodes	002	Medium
Docker-Proxy Role Is Too Broad	005	Medium
Sensitive Docker Ports Accessible to Containers	003	Low
AWS Metadata API Exposed to Containers	004	Low

Finding Privileged IAM Role Passed to CloudFormation Stack

Risk Medium Impact: High, Exploitability: Medium

Identifier NCC-DOCK_AWS_2017-001

Category Access Controls

Location https://docs.docker.com/docker-for-aws/iam-permissions/

Impact Any user granted the permission to use the Docker-recommended and highly permissive CloudFormation role (e.g. given create and execute change sets) by extension is granted that CloudFormation's permissions and as a result may be able to escalate privileges

Description By default, CloudFormation stacks inherit the current calling user's access permissions, which prevents the formation from performing any operation outside the user's permissions. If the user assigns a service role to CloudFormation stack, CloudFormation will use that service role's permissions instead of the current user's. As a result, any user who is granted the permissions to access and update that formation will also inherit any permissions granted to the formation's service role, even if the user did not originally have those permissions.

Docker for AWS recommends that users create a service role for the Docker CloudFormation stack.[12] The recommended set of permissions is too permissive, allowing the role to perform, among other things, potentially dangerous commands, such as:

- Create new roles in the AWS account.
- Create and destroy any EC2 instance in the AWS account.
- Create and destroy any VPC in the account.
- Attach any created role to any EC2 instance.

Additionally, the current Docker-recommended permissions are not constrained to the Docker-created AWS VPC or any specific domain. Any permission assigned to the service role applies across the entire AWS account. Furthermore, some of the permissions are account-wide privileges and cannot be constrained to a VPC, such as the creating roles permissions.

Recommendation NCC Group recommends that Docker provide users with the ability to constrain the CloudFormation service role. In order for this to be feasible, the following configuration options need to be added to the CloudFormation stack setup and made available to users:

1. The ability to create a new stack prior to running the formation and constrain the resources of all "cloudformation:" operations to that Stack.

2. The ability to create a new VPC prior to running the formation and constrain the resources of all "ec2:" operations to only that VPC.

3. The ability to create a new AUTOSCALING group prior to running the formation and constrain the resources of all "autoscaling:" operations to the group.

4. The ability to create a new DYNAMODB table prior to running the formation and constrain the resources of all "dynamodb:" operation to that table.

5. The ability to create a new LOG group prior to running the formation and constrain the resources of all "log:" operations to that group.

6. The ability to create a new a SQS queue prior to running the formation and constrain the

[12] https://docs.docker.com/docker-for-aws/iam-permissions/

resources of all "sqs:" operations to that queue.

7. The ability to create a new ELASTIC load balancing group prior to running the formation and constrain the resources of all "elasticloadbalancing:" operations to that group.

Furthermore, the roles needed by Docker worker and manager nodes should be created prior to launching the CloudFormation and the set of IAM permissions should be reduced to just "iam:AddRoleToInstanceProfile" and constrained to the created role and VPC. Finally, the "elasticfilesystem:*" permissions should be removed as it currently has limited resource constraining capabilities.

Finding Docker-Proxy Role Passed to Worker Nodes

Risk **Medium** Impact: Medium, Exploitability: Medium

Identifier NCC-DOCK_AWS_2017-002

Category Access Controls

Impact Users with access to a Docker worker node may perform a number of AWS API calls across the entire AWS account, which may disclose and/or affect resources not created by Docker for AWS.

Description The `Docker-Proxy` role allows for the following operations to be performed across the entire AWS account by worker nodes:

- All Elastic Load Balancing operations, using wildcard on `elasticloadbalancing:*`
- All AWS autoscaling operations, using wildcard on `autoscaling:*`
- Describe any EC2 Instance
- Describe any VPC
- Create Log Streams
- Put Log Stream events

NCC Group believes that the listed capabilities should not be added to the role, since any container running on the worker node has access to the AWS metadata service as described in finding NCC-DOCK_AWS_2017-004 on page 14, there is the potential that attackers or internal low-privileged users could make calls to the Metadata URI to retrieve the instance's role's credentials and use these to make AWS API calls from a compromised container.

NOTE: Docker provided NCC Group with the updated code that implements the recommendation outlined below. NCC Group did not verify if it is correctly being deployed, as the final version of the code was still being developed.

Recommendation Do not pass the `Docker-Proxy` role to worker nodes as only manager nodes need to use this role to access the AWS APIs.

Finding Docker-Proxy Role Is Too Broad

Risk **Medium** Impact: High, Exploitability: Medium

Identifier NCC-DOCK_AWS_2017-005

Category Configuration

Impact Any user with access to the manager and/or worker nodes can issue sensitive commands across the entire AWS account and is not limited to only the Docker VPC. This includes all Elastic Load Balancer and AWS autoscaling operations as well as retrieving details on any EC2 instance or VPC.

Description The `Docker-Proxy` role assigned to worker nodes allows several potentially sensitive AWS operations to be performed across the entire AWS account without restriction. The `Docker-Proxy` role allows for the following operations to be performed by manager and worker nodes:

- All Elastic Load Balancing operations, using wildcard on `elasticloadbalancing:*`
- All AWS autoscaling operations, using wildcard on `autoscaling:*`
- Describe any EC2 Instance
- Describe any VPC
- Create Log Streams
- Put Log Stream events

NCC Group believes that the listed capabilities should be restricted to only the current VPC created by Docker for AWS and not to the whole AWS account, as it does currently.

NOTE: Docker provided NCC Group with the updated code that implements the recommendation outlined below. NCC Group did not verify if it is correctly being deployed, as the final version of the code was still being developed.

Recommendation The current `Docker-Proxy` role should restrict all allowed operations to only the Docker-created VPC and resources.

Finding: Sensitive Docker Ports Accessible to Containers

Risk Low Impact: Low, Exploitability: High

Identifier NCC-DOCK_AWS_2017-003

Category Access Controls

Impact An attacker with access to a container within a Docker for AWS installation can contact the diagnostics, gossip, VXLAN, and Docker swarm mode join ports on the Docker swarm mode hosts.

Description To perform the work of a standard Docker swarm mode setup, Docker swarm mode hosts expose a number of ports for communicating LAN traffic, cluster join information, and other details. These ports are accessible to any container running in the cluster, as no internal firewalling is performed between the cluster hosts and the containers running on them.

From discussions with the Docker team, the team has indicated that the gossip, VXLAN, and Docker swarm mode join ports require authentication credentials that individual containers would not have.[13] As a result, this issue is a matter of an increased attack surface being exposed, not sensitive data or network access. The diagnostics port does not expose sensitive data to users, although it can be used to trigger diagnostic reporting, which may be a potential denial of service vector. Docker intends to rate-limit this endpoint in the near future, which will reduce this risk.

Reproduction Steps

1. Set up a Docker for AWS cluster.
2. Log in to the management container.
3. Run a simple container with a shell: `docker run --network=bridge -it ubuntu /bin/bash` (`--network=bridge` is the default, we specify it only to be explicit that this is not the "host" network)
4. Install nmap: `apt update && apt install -y nmap`
5. Run `nmap -p 2377,7946,44554, 10.0.0.4-6` (or different IP addresses if your cluster manager and workers have different internal IP addresses) and note that ports `7946` (the gossip port) and `44554` (diagnostics) are open on all hosts, and port `2377` (Docker swarm mode join) is open on the swarm manager.
6. Run `nmap -p 4789,1234 -sU 10.0.0.4-6` (or different IP addresses if your cluster manager and workers have different internal IP addresses) and note that UDP port `4789` (the VXLAN port) is open. (Port `1234` is included as an example of an actually closed port, to differentiate the `open|filtered` status of port `4789`.)

Recommendation Use a host-based firewall on each Docker swarm mode host to prohibit traffic from container bridges to the swarm network.

[13] https://docs.docker.com/engine/userguide/networking/overlay-security-model/

Finding: AWS Metadata API Exposed to Containers

Risk Low Impact: Low, Exploitability: High

Identifier NCC-DOCK_AWS_2017-004

Category Access Controls

Impact An attacker with the ability to issue HTTP commands on the container, can call the AWS metadata API, which contains potentially sensitive host information, such as, but not limited to, hostname, kernel version, internal IP address, and OpenSSH public keys. Furthermore, if any IAM role is associated with the instance, the IAM role information as well as a set of temporary credentials to access that role may be obtained.

Description AWS provides a metadata retrieval API on a link-local address that is available via HTTP requests to http://169.254.169.254/latest/meta-data/ to any instance running on EC2. The metadata API provides instance metadata such as hostname, IP address, AWS internal host ID, and more.[14] Furthermore, if an AWS IAM role is assigned to an instance, the metadata API can be used to gain access to a set of temporary credentials that may be used to assume that role.[15]

Docker containers running on Docker for AWS have full access to the AWS instance metadata API. In addition, Docker chooses to enforce instance roles, as described in finding NCC-DOCK_AWS_2017-002 on page 11. Furthermore, during NCC Group's testing no sensitive data was being stored in the EC2 instance user data, limiting the amount of sensitive data potentially being exposed.

Reproduction Steps

1. Set up a Docker for AWS cluster.
2. Log in to the management container.
3. Run a simple container with a shell: `docker run --network=bridge -it ubuntu /bin/bash` (`--network=bridge` is the default, we specify it only to be explicit that this is not the "host" network)
4. Install `curl`: `apt update && apt install -y curl`
5. Run the following command: `curl 'http://169.254.169.254/latest/meta-data/'`

Recommendation Use a host-based firewall on all Docker hosts to prohibit access to `169.254.169.254`. If access is required by the container, implement a push mechanism from the worker node to the container, pushing only the required data. Use caution when extending the access granted to EC2 instances via instance roles as any container breakout would give an attacker that level of access into your infrastructure.

[14] https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-instance-metadata.html

[15] https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/iam-roles-for-amazon-ec2.html#instance-metadata-security-credentials

Appendix A: Finding Field Definitions

The following sections describe the risk rating and category assigned to issues NCC Group identified.

Risk Scale

NCC Group uses a composite risk score that takes into account the severity of the risk, application's exposure and user population, technical difficulty of exploitation, and other factors. The risk rating is NCC Group's recommended prioritization for addressing findings. Every organization has a different risk sensitivity, so to some extent these recommendations are more relative than absolute guidelines.

Overall Risk

Overall risk reflects NCC Group's estimation of the risk that a finding poses to the target system or systems. It takes into account the impact of the finding, the difficulty of exploitation, and any other relevant factors.

Critical Implies an immediate, easily accessible threat of total compromise.

High Implies an immediate threat of system compromise, or an easily accessible threat of large-scale breach.

Medium A difficult to exploit threat of large-scale breach, or easy compromise of a small portion of the application.

Low Implies a relatively minor threat to the application.

Informational No immediate threat to the application. May provide suggestions for application improvement, functional issues with the application, or conditions that could later lead to an exploitable finding.

Impact

Impact reflects the effects that successful exploitation upon the target system or systems. It takes into account potential losses of confidentiality, integrity and availability, as well as potential reputational losses.

High Attackers can read or modify all data in a system, execute arbitrary code on the system, or escalate their privileges to superuser level.

Medium Attackers can read or modify some unauthorized data on a system, deny access to that system, or gain significant internal technical information.

Low Attackers can gain small amounts of unauthorized information or slightly degrade system performance. May have a negative public perception of security.

Exploitability

Exploitability reflects the ease with which attackers may exploit a finding. It takes into account the level of access required, availability of exploitation information, requirements relating to social engineering, race conditions, brute forcing, etc, and other impediments to exploitation.

High Attackers can unilaterally exploit the finding without special permissions or significant roadblocks.

Medium Attackers would need to leverage a third party, gain non-public information, exploit a race condition, already have privileged access, or otherwise overcome moderate hurdles in order to exploit the finding.

Low Exploitation requires implausible social engineering, a difficult race condition, guessing difficult-to-guess data, or is otherwise unlikely.

Category

NCC Group categorizes findings based on the security area to which those findings belong. This can help organizations identify gaps in secure development, deployment, patching, etc.

Access Controls Related to authorization of users, and assessment of rights.

Auditing and Logging Related to auditing of actions, or logging of problems.

Authentication Related to the identification of users.

Configuration Related to security configurations of servers, devices, or software.

Cryptography Related to mathematical protections for data.

Data Exposure Related to unintended exposure of sensitive information.

Data Validation Related to improper reliance on the structure or values of data.

Denial of Service Related to causing system failure.

Error Reporting Related to the reporting of error conditions in a secure fashion.

Patching Related to keeping software up to date.

Session Management Related to the identification of authenticated users.

Timing Related to race conditions, locking, or order of operations.