Security is incredibly important at PagerDuty. As an operations performance platform for some of the world’s largest companies, PagerDuty has a responsibility to its customers to provide the highest possible levels of security at all times.

Yet, due to the PagerDuty platform’s complexity — PagerDuty, as a cloud service, is highly dependent on the performance of a number of web applications — security optimization remains a constant challenge.

Here we explore some of the methods the PagerDuty Operations Engineering team has designed for security. Consider it a “best practice” guide to operations security in the age of web apps.

The Right Philosophy & Role Assignments

PagerDuty treats security conventions as a foundational piece of its security methodology. To the extent that there is a list of conventions, it is not written in stone. Yet establishing a philosophy around security helps PagerDuty’s engineers understand where trade-offs are being made (and speeds decision making). It also makes it easy for new engineers to quickly understand why things are set up the way they are.

One key convention is to secure everything by default. That means that disabling any security service has to be done via an override or exception rule, so as to enforce consistency across PagerDuty’s development, test and production environments.

This convention is particularly important in test and production. As tempting as it may be to poke a hole in the local firewall or to disable SSL when connecting to MySQL, these types of security changes are not appropriate in either the test or production environments. Setting tools to automatically “do the right thing” keeps engineers honest. Also, by having this kind of consistency, security-related issues are discovered earlier in the development cycle.

Another important convention relates to the level of protection on the network. In a security model where only the edge of the network is protected, an attacker that is able to penetrate that single layer will gain access to everything. This is one reason why PagerDuty follows a
least-privilege permissions model, in which engineers only have access to the servers they need to get their job done.

Adding a new user to PagerDuty’s infrastructure means defining the groups to which this user belongs. That comes into play when a new group of servers is added: Only certain user groups will have access to the new server group. (Group information is stored in JSON config files and is in version control, which, as a bonus, helps make code review simpler and safer.)

Platform and Provider Agnosticism

PagerDuty might be evangelical about security optimization, but agnosticism is the name of the game when it comes to platforms and vendors.

All of PagerDuty’s security tooling is based on commonly available Linux tools or installable packages. On the architecture side, the PagerDuty platform leverages multiple hosting providers—and the infrastructure behind it is fully location-agnostic, which means it can be adjusted on the fly as real-time needs and provider availability warrant.

While a platform-agnostic approach limits the extent to which certain tools—such as security groups, private clouds and rescue consoles—may be used (since those tools are provider-specific), the benefits of agnosticism outweigh the drawbacks. Not only does having multiple providers provide fewer points of failure, but depending on provider-specific tools can lead to vendor lock-in.

In terms of network design, PagerDuty has implemented a point-to-point encryption model based on IPSec in transport mode. This enables all traffic between specified nodes on the network to be encrypted, regardless of where the node is located and what other nodes it is talking to.

Why use this methodology instead of Virtual Private Networks (VPNs) or encryption on a per-app or service basis?

A typical VPN implementation with dedicated gateways at each region would have had a number of issues:

- **Almost single-point-of-failure.** Any time a regional gateway server goes away, there is a scramble to address the reduced capacity.

- **Cost and scalability.** PagerDuty’s reliance on virtual machines versus dedicated networking hardware would produce very large instance sizes to encrypt and decrypt traffic for the servers behind them. Conventional VPN gateways would likely struggle to scale with these traffic spikes.

- **Latency.** The goal is to have as few hops as possible when connecting to non-local servers.
• **Within-region encryption.** All cross-region WAN traffic is being encrypted, but within each region, there are multiple datacenters where the data is not encrypted. It has recently come to light that even dedicated connections between datacenters have been compromised, so this is an area of concern.

Encrypting on an app-by-app or service-by-service basis, meanwhile, introduces the following problems (at least when it is used exclusively):

• **It’s easy to forget.** While security is part of everyone’s job at a company, many times people will forget to enable the appropriate security settings.

• **Each app/service has a slightly different way of encrypting data.** While most connection libraries support SSL, it can be implemented differently each time. Moreover, this means that anytime a new service is added, a different encryption methodology must be addressed.

Per-app or per-service encryption can make sense in certain circumstances. Forcing MySQL to only allow SSL connections or making sure that Cassandra uses internode encryption, to offer two examples, are appropriate within the PagerDuty security framework.

Ultimately, though, point-to-point encryption is the most logical methodology. Point-to-point encryption offers:

• **Decentralized encryption.** Instead of relying on critical VPN gateways, each node can handle its own encryption (removing single points of failure).

• **Scalability.** Since relationships are only calculated for the nodes that a single node needs to talk to (as opposed to every node), the overhead of the encryption is quite low.

• **Efficiency.** Most modern chipsets ship with dedicated AES hardware. Additionally, since the traffic is encrypted as well as compressed, network throughput is only minimally impacted.

• **Within-datacenter encryption.** Sending traffic over dedicated links within or across datacenters is generally secure, but recent events have raised the specter of security breakdowns in these kinds of connections. Point-to-point encryption provides a better alternative.

• **One less dependency on NAT.** As more networks support IPv6 and a global address space, the private address space provided by VPNs will have to be re-done. The point-to-point model easily supports a global address space.
Secure by Design

PagerDuty's operations engineers have spent an enormous amount of time designing and architecting the service to withstand failures across servers, datacenters, regions, providers, external dependencies and many other factors. But challenges persist.

As a cloud service, PagerDuty deploys infrastructure to hosting providers (whose networks are out of PagerDuty’s control). Infrastructure deployment is complicated by the fact that it crosses multiple regions, which forces a significant amount of data traffic over the WAN (a.k.a. the internet). This introduces the challenges of packet loss and high latency—as well as the possibility that intruders may try to eavesdrop. With all these factors in mind, data is encrypted in flight—under the assumption that it is flowing through low-visibility networks.

On the application side, meanwhile, PagerDuty bucks convention to an extent. It’s common to have single sources of truth for both access control and authorization: LDAP servers that store access control settings or networking hardware that rejects incoming connections from blacklisted IP addresses, for example. But PagerDuty treats these tools as single points of failure (imagine a single VPN appliance going down and taking out an entire datacenter).

To keep failure at bay, PagerDuty’s security policies are managed centrally. Rulesets are then pushed to all the nodes in the network. Instead of relying on single sources of truth for both policy management and enforcement, the enforcement pieces are split out and distributed to the individual nodes in the network. Thus, when a change is introduced into the network, the single source of truth updates the ruleset and then pushes out the changes to each node.

That provides the groundwork for a robust system. But what about actually managing threats on an ongoing basis? One essential element here is dynamic firewalls.

These firewalls come into play when a new cluster of servers is defined in Chef. A ruleset for the firewalls is set up to define the group that the servers are in (and establish which other groups can talk to the inbound port on this new group). Then, Chef can create entire IPTables chains automatically, open the appropriate ports for the appropriate servers, drop all other traffic, and push this ruleset out to each server. Each time a server is added or removed, the ruleset is calculated for each server and then pushed out again.

The benefits from this approach are as follows:

- **Network partitions** may be created as needed to ensure the dev, test, and production environments cannot talk to each other.

- **Individual servers may be isolated** when the real-time team needs to practice attacking them.
- **Server communication is easy to track**, since all of the inbound rules have to be defined upfront.

- **IPTables are simple and straightforward to use.** If there is a firewall problem, every engineer understands how to manipulate the firewall and deploy a fix.

- **There is no single-point-of-failure network device.** If a single server goes down or something more catastrophic happens, the rest of the system will continue to operate in a secure fashion.

Chef is also useful for enforcing role-based access, as discussed in the section entitled “The Right Philosophy and Role Assignments” above. Chef is employed in concert with straightforward Linux users/groups to build out access controls.
Monitor and Validate

While all of the above serves to provide a robust security architecture, it’s important that security measures are constantly validated to ensure that they’re doing what they are supposed to do.

Some of the monitoring and alerting tactics in use at PagerDuty include:

- **Port Availability Monitoring**
  Dynamic firewalls (discussed in the “Secure by Design” section above) are used to maintain a list of ports that should be open or closed to the world. This information is held in the Chef server, which acts as a central repository for checking which ports should be open or closed on which server. If one of these checks (run by a framework called gauntlt) fails, PagerDuty’s engineers receive an alert.

- **Centralized Logging and Analysis**
  Because attackers can shut down any logging to hide their tracks, PagerDuty centralizes logging with Sumologic. Shipping logs somewhere else and setting up pattern alerts on them both maximizes log security and accelerates resolution time for an event.

- **Time Series Data for Security**
  As described in the “Platform and Provider Agnosticism” section above, IPSec is the preferred encryption methodology at PagerDuty. IPSec provides limited health monitoring, however, outside of mining syslog. PagerDuty’s engineers addressed this by writing scripts that pull down IPSec statistics, applying counters and gauges on them and sending them to statsd.

- **Active Response**
  PagerDuty is still early in its active-response implementation, but active response is playing an ever-larger role in infrastructure security. Tools that can take action without any input from team members are a must-have in large infrastructure deployments; they make ongoing management easier, too, by limiting the extent to which engineers have to react to security incidents.

  One active-response tool in use is DenyHosts, which has been deployed to every server in PagerDuty’s infrastructure. If a non-existent user tries to login or if there is another brute force attack, the tool will block the offender’s IP address. Since DenyHosts was configured in 2013, 1,085 unique IP addresses from have been blocked.

  Another useful tool is OSSEC. This open-source intrusion detection system helps to detect strange behavior on PagerDuty’s servers by continuously analyzing critical log files and directories for anomalous changes. OSSEC has different alert “levels”: Low and medium-level ones will send out an email, while high-level one will create a PagerDuty incident so an operations engineer can immediately respond to the problem.
What about proactively checking the robustness of the PagerDuty security framework? Some companies will do quarterly penetration testing, but in PagerDuty’s constantly changing application environment, that’s too slow. Actively monitoring and alerting on changes means mistakes (e.g., an engineer opening the wrong network port on a server) or malicious behavior are caught quickly.

Worth noting is that all of these monitoring tactics are enhanced when implemented in concert with the PagerDuty service itself, which centralizes alerting to keep the right teams in the loop. By relying on our platform for alerting, PagerDuty’s Operations Engineering team stays on top of system changes or application issues as they come up. That, in turn, enables the provision of maximum uptime and the highest level of service to PagerDuty’s customers—not to mention the best possible security.