Security discussion of the Heroku platform

Charlie Egan

1 Introduction

This report covers the company Heroku in detail from a risk and security standpoint. Heroku offer a web application hosting platform-as-a-service with support for a range of languages. In 2010 the company became Salesforce subsidiary.

Their service for hosting web applications has a strong usability focus and is marketed at developers, not operations professionals. The service allows users to deploy their code to a publicly visible subdomain for free. Heroku makes scaling applications easier by providing a web interface to adjust allocated resources. A library of add-ons enables users to extend their applications utilising 3rd party services not included by default. These are the main selling points of the platform. For a single team to architect such a flexible system is, operationally, very challenging. Heroku is often seen as a cost effective alternative. [10]

The company employs a freemium business model. Users can deploy free applications, but as their application scales they incur expenses. Computing resources are prorated to the second, premium add-ons such as Elasticsearch have a fixed monthly fee. Applications on the free tier can only be active for 18 hours daily and sleep after 30 minutes of inactivity. They are also limited to 10,000 rows in the included database. Heroku abstract computing resources using *Dynos*, these are portable containers that can be hosted anywhere on the platform to run a user's process.

Users of the platform are predominantly developers ranging from learners working on side projects to startup teams. While the platform is also commonly used for 'throw-away' staging environments it also hosts 17 Quantcast top 10k websites [4] and a wide range of high profile startups such as the automation platform IFTTT and Product Hunt [14].

Security at Heroku is handled by a dedicated team, they also employ a Chief Information Security Officer (CISO) [28]. Their CISO is responsible for "application security, security operations, compliance, and external security relations" [7]. The current *Director of Platform Security* is Jacob Kaplan-Moss [5]. Contact information to report vulnerabilities is made clear in their security policy [31]. The company also publishes a "Security Researcher Hall of Fame" list [27].

Heroku must comply with local data protection laws in all regions in which they offer services. Their data protection procedures for deletion and physical security are detailed in Salesforce documentation [50]. In the UK, Heroku are liable to pay compensation to any individuals that suffer from the result of Heroku's non-compliance [47]. Heroku must also comply with The Payment Card Industry Data Security Standard (PCI DSS), their infrastructure provider is PCI compliant, Heroku also use the compliant third party payment processor *Braintree* for taking their own payments [25].

This report is an in-depth analysis and discussion of both the Heroku website and connected platform from a security perspective. Security related properties of the system will be detailed and related vulnerabilities highlighted and commented on.

Heroku is a highly connected service. The platform runs on Amazon's infrastructure as a service, AWS - this report will not cover this outside of its relationship with Heroku. Customer applications running on the platform are also at risk from security vulnerabilities. This report will only discuss such applications, and any related addons, where they interact with the platform - potential vulnerabilities of these applications is only mentioned in passing.

2 Discussion

2.1 Asset & System Identification

2.1.1 System Architecture

The following description summarises the publicly available information on the platform's architecture and accompanies Figure 2 on page 10.

The first points of contact are a series of DNS servers which route user application subdomains and custom domains to nginx proxies.

Connections made to the proxies, retrieved from the DNS lookup, are routed to a Varnish HTTP cache layer [44]. In the default event of a cache miss, the request is forwarded to the *Routing Mesh* [20]. This is a custom Erlang component of the platform [44] and is responsible for forwarding requests to the correct application server (Dyno) [21].

Dynos are the company's unit of computation and sand boxing implementation. They are hosted on the *Dyno Grid* [49]. This is an array of 'Railgun' server instances, each hosting around 60 Dynos and deployed to an AWS EC2 instance.

Dynos are built and deployed to a Railgun instance after a git push is completed. Repositories trigger pre-receive hooks to initiate the deployment process when users push [29]. Next the application code is 'compiled' into a slug using commands defined in an open source 'buildpack' [13]. The result is a SquashFS, read-only slug that can be used to provision the application's Dynos [16]. Addons and databases are also provisioned on Dynos and automatically backed up to the AWS Elastic Block Store. [26]

While this represents the core functionality, there are additional undocumented services that administer the platform. These will consist of a web application that hosts the *heroku.com* and implements the features of the user dashboard. This application needs make changes to the platform as well as a customer database. It is also likely connected to a number of other internal services for analytics and logging.

2.1.2 Asset Identification

The Heroku business model is built on the following classes of information assets.

The git server that users push code to during deployment is one of the more sensitive on the platform. Eight languages are supported and, when deploying, users must upload their code in full - even for compiled languages (Go) when a binary could be compiled elsewhere. [18]. This git server contains the business logic and algorithms of every customer. Representing a large repository of company secrets, makes this a particularly sensitive asset. The system is also highly critical, during downtime users would be unable to redeploy applications to fix bugs and release updates. This service has been interrupted for 85 minutes this year to date [22] [23]. To ensure confidentiality, users communicate using ssh with an RSA or DSA key [12], each user on the platform is only given access rights to the git remotes associated with their applications.

Application Dynos also contain source code and additionally environment variables to connect with additional services such as databases, banking software or social media accounts. A Dyno could also be a database supporting an application - in this case it might include private communications, cached bank details or information that may enable an attacker to infiltrate accounts elsewhere. Dyno downtime also effects their production uptime statistic, so both confidentiality and availability are critical. Heroku publish uptime statistics [19] as well as a comprehensive incident report [17]. Downtime prevention is discussed in the Countermeasures section.

The DNS servers and the records they store are also highly critical, while the information is public, it is important that it not be altered by an unauthorised party. This would allow traffic to be routed to a fraudulent application. The integrity of the information this service provides is paramount and can only be altered by a user over ssh or HTTPS.

Finally, there are Heroku's own datastores which store customer, billing and application data. This information, and its backups, are critical as they enable recovery after a service outage. Customer banking details are highly sensitive and the availability of application information is critical to the functioning of the platform. In a 2010 press release [51], Heroku announced they are using Splunk as a monitoring solution to track and maintain integrity in these core systems.

2.2 Risk Identification

2.2.1 Threat Analysis

Threats with the highest impact on Heroku's system are predominantly from human agents exploiting human vulnerabilities. Being built on AWS means means that some physical security risks are transferred. Customers also take responsibility for their own application security - though the platform offers a good baseline [24]. Unpredictable and deliberate threats pose a far greater risk to Heroku and their customers. Hacking attempts are not publicised unless service is interrupted, making it hard to gauge their frequency.

Hackers are the most prevalent threat agent type and pose the greatest risk. Infiltrating the platform has the potential to yield valuable information about customer applications and their users. Details could include banking credentials, copyrighted material or prerequisite information for follow-up attacks.

Heroku also has a good security record and could be seen as a prize target among hackers. Political content is not restricted on the platform so controversial sites could also draw attention [11]. Heroku does however implement DDoS mitigation techniques and publicises this to deter hackers [15]. Heroku maintains a public record of all service outages, since 2009 the service has been interrupted by DDoS attacks 3 times totaling 32 hours. No other types of attack caused outages. [6]

Employees are another threat agent. While customers operate in sandboxed environments, a disgruntled or incompetent employee could be extremely damaging. With greater privileges and system knowledge they represent a key vulnerability.

2.2.2 Identification of Vulnerabilities and Exploits

Heroku is under a great deal of scrutiny and while vulnerabilities must exist, and are reported on their Hall of Fame, they appear to be rare. Heroku were effected by events such Heartblead [8] and continue to be susceptible to similar bugs. Heroku claim their systems are kept up-to-date without the need for user interaction [30] - however, there were disruptions in the production environment during Heartblead.

Enabling password resets via email makes customer email accounts a potential weakness. Heroku has no control over the security measures at a customers' email provider. Gmail session cookies, for example, expire after two years. Using this weakness, attackers and could utilise Cross Site Scripting to collect cookies and gain email access. Social Engineering tactics could also be employed as in the case of journalist Mat Honan in 2012 [45].

It is clear that customer and employee mistakes represent a vulnerability. Last year the company introduced 2 factor authentication [9] - this makes attacks on user accounts more likely to be made using social engineering. AWS, Heroku's infrastructure provider, has been subject to attacks of this type in the past [3].

Please see Figure 1 on page 9 for a threat tree depicting unauthorised account access.

2.3 Identification of Existing Countermeasures

2.3.1 Access Control

Authenticated customers manage their applications and account via the user dashboard. They can perform the following actions that are potentially destructive.

- Addition and Revocation of SSH keys
- Password Update
- Billing History and Details
- Application Ownership Transfer
- Application Creation, Scaling & Deletion
- Application DNS Configuration

None of these actions require additional validation once a user has logged in. An application could be scaled to incur a large bill, transferred to another owner or simply shutdown - however some actions, such as adding a new SSH key, do trigger a notification email. Bank details are not exposed in full but they can be updated.

There are no features in the dashboard that aggregate customer preferences, even add-ons cannot be sorted by popularity. When creating a new app you are prevented from using an existing name. While this does enable customers to discover the applications of others, each site is already public and customers know app URIs are not secret.

Customers can add collaborators to their applications, these users are able to administer the application with the exception of adjusting paid add-ons, deleting or transferring the app and viewing invoices [33]. Heroku is aware any user communication or aggregation represents a potential weakness and limits such interactions. This enables them to offer these important features without introducing more risk than strictly necessary.

At a Dyno level, within the platform, user processes are strictly sandboxed. Dynos are deployed using *LXC* (an application container implementation) to guarantee isolation, even on a multi-tenant host - which is itself virtualised [38]. While in theory the platform makes such a precaution possible, the claims are brought into question by the recent Digtial Ocean Private Networking flaw [52].

2.3.2 Identification & Authentication

Users are authenticated using an email-password combination and, as of last year, customers are also able to opt-in to mobile two factor authentication [9]. When using the Heroku Toolbelt to interact with the platform from the command line they are able to use ssh or HTTPS. Each user has a unique email address. Users must create their own password after validating they have access to the supplied email address. Passwords must be 8 or more characters in length and contain at least two of the following: case-insensitive letter, number or symbol. When a user forgets their password they are prompted to enter their email and are sent a reset link without any additional checks.

Session cookies are used to identify users. The dashboard is built using the client-side MVC framework Ember.js [32]. User details are held in a local storage object which expires after a few hours. The object contains the authentication method (OAuth 2.0), an access token & refresh token for the session, token type (Bearer), expiry information, an obfuscated user id, email address and a session nonce. Refresh tokens are used by the client when requesting a new access token [1], nonces are used to ensure to protection against repeated data submission and man-in-the-middle attacks [46]. The session cookies must be accepted (Privacy Policy) to use the service [41].

It would be hard to introduce an additional mandatory factor, such as the unpopular peripheral authenticator [48], to the login process without negatively impacting usability.

2.3.3 Further Countermeasures

Besides the controls surrounding the user dashboard and applications, Heroku employ a number of further countermeasures to protect the platform as a whole.

Internal Firewalls limit connections within the platform to a strict set of required ports. Open ports on each instance are limited to those required to run its service. ‘Security Groups’ are used to define classes for the specific purpose of each instance. Customer database Dynos, application Dynos and Heroku’s management infrastructure all run separate network security groups [43]. Additional firewalls within network instances restrict connections over the virtual network interface to `localhost` further isolating customer code and reducing risk. [37]

While Port Scanning is not automatically blocked it is prohibited on the platform and either internal or external IPs doing so are blacklisted [40]. Packet Sniffing is also disabled by Dyno hosts, connections between Dynos on the platform are also encrypted. [42]

All components are kept at the latest patch version. Information about the latest security patches comes from security assessments, mailing lists and vulnerability reporting services. In the event of a new platform vulnerability the risk is assessed before being passed to a team to resolve the issue - patches are validated by the security team. [43]

External security assessments and penetration tests are also contracted. These focus on the OWASP vulnerabilities and ensuring isolation is correctly enforced. The assessments cover all components of the platform. [43]

Heroku caries out background checks on all new employees [35]. AWS employees are granted data center access per-project, access is immediately revoked when the project finishes.

AWS data centers have fire detection and suppression, an uninterruptible power supply and are closely monitored for hardware defects [36]. They are also ISO 27001 and FISMA certified. [39]

Customer application data is regularly backed up. Backups are used to automatically redeploy user applications in the event of an outage. [34]

2.3.4 Imagined Countermeasures

A hosting company in control of valuable data and critical services needs to be transparent about security policies and countermeasures. However, aside from those outlined above, the following also likely exist.

The company is likely to have an internal password policy that details password requirements for staff. Constraints on required length can be enforced automatically but ensuring regular resets and password sharing rules requires internal agreement. Creating a corporate culture that that encourages security awareness and good practices reduces the probability of a wide range of attack types.

An employee termination policy also likely been established. This will typically involve revoking & resetting their credentials, recalling equipment and revoking access to internal discussion areas. This helps limit the damage of a disgruntled employee and reduces the chance that an ex-corporate computer is resold without the company's knowledge.

It is also probable that there is some form of monitoring tool built specifically for the Heroku platform. Resolution of threats such as DDoS and hardware failure are hard to automate entirely. A member of staff is likely 'on support' 24 hours a day to monitor the load and status of various services. Human supervision of an expressive monitoring tool reduces the chance of a fault or attack going unnoticed. While responding to the issue quickly does not reduce the likelihood, it will help significantly reduce impact.

Employee contracts will also have a clause that restricts them from disclosing details of the platform's architecture and likely applies to employees after the termination. This acts as a deterrent and gives the company a legal fallback.

2.4 Determining Risk

This section is documented in the risk matrix and register in Appendices 1 and 2 on pages 7 and 8.

2.5 Control Recommendations

In response to the sample outlined in the risk register, I propose the following controls, ordered by priority, to better mitigate risk.

The minimum bar for customer authentication is low - especially given the potential value of a customer's application. A new policy could either prompt or require customers running high profile sites to enable two factor authentication. This introduces an additional barrier for an attacker of higher profile targets. Heroku could also transfer the risk to an OAuth provider, though they would loose the option to conditionally enforce restrictions such as this that are based on customer types. This is a lower priority risk but, when it is trivial to determine a sites' hosting provider, Heroku must seek to minimise the impact of becoming a target via a customer. Additional verification could also be required for destructive actions.

Limiting knowledge of vulnerabilities within the company to those implementing a fix reduces the chance of vulnerability information being leaked. This is a simple way to minimise these high impact risks.

Heroku could also improve the default level of security for customer applications by offering SSL as standard. Other PaaS providers such as Cloudfare provide this feature at no additional cost.

Heroku might also consider introducing an OWASP 'vulnerability scanner' to automate detection of common flaws in their customers applications. This information could be presented as notifications and provide actionable feedback and would help avoid bad publicity for an attack on a customers' application. Making customers aware of their security responsibilities is a good way to increase the security the whole platform. While customer vulnerabilities have a high likelihood they have limited impact due to sandboxing.

3 Critique

Risk management needs to be evaluated differently for applications running on cloud infrastructure. A key change is that at any given time, the infrastructure in use by the platform must first be determined. As usage fluctuates, the system resizes accordingly making it impossible to talk about 'a server' as a distinct asset.

Instead components of the system must be grouped into classes of comparable units, as above in the *Asset Identification* section. Grouping components like this makes it possible to generalise - e.g. all Dynos are one point minor version behind. Targets within the system fall into a group and the associated risk can still be managed. Of the risk management process, only the definition of asset need be adjusted, threat agents and vulnerabilities remain the same.

Assigning qualitative risk in a highly interconnected system is likely to fall short when applied in practice. Rather than a specificity problem I see this as a conditional one, additional connections make calculating the likeliness and impact of event chains much harder. More components need to be accounted for making it more likely something is overlooked or misrepresented.

In a system like that of the Heroku platform, one in a constant state of flux by design, analysing threat vulnerability pairs is made much harder. Making connections from the threat agent's point of entry to the target asset is often unclear. For example, an attacker could access the database of the customer's application via the dashboard, ssh, a vulnerability in the platform itself or via a flaw in the customer's own site. There comes a point when the number of interconnected parts makes it impossible to accurately to calculate the conditional probabilities for vulnerability likeliness.

There are a number of problems in applying risk management to such a system. The complexity introduced by having different companies running parts of the system makes the risk evaluation much harder. I see it as three parties, Heroku, its customers and the infrastructure provider. In a system like this where each, in terms of communication, are (relatively) disconnected, it becomes a case of working to best limit their own risk in their part of the system. This makes traditional exhaustive risk management techniques harder to apply. [2]

Risk management is also based on valuing the system's assets. In a traditional system this was done by looking at networked devices and aggregating replacement and data loss costs. On a cloud system where a physical machine is running many servers, of which the value of each is unknown, asset evaluation becomes impossible. There is also the residual issue of attempting to quantify the cost of lost opportunity. Replacing hardware or spawning another instance is cheap, estimating the value of the work lost by downtime on systems is very challenging to do accurately. [53]

However, I think that the risk management still brings value to a cloud PaaS provider. It opens up the discussion about risks, helps educate employees, and in turn customers about threat agents and common vulnerabilities. Better security awareness greatly benefits Heroku and is likely the most effective means the company has to improve the security of their platform as a whole.

4 Appendix 1. Risk Matrix

Threat vulnerability pairs:

1. Terminated Employees - Un-patched Vulnerability
2. Digruntled Employee - Avoidable Logging System
3. Hardware Failure - Incomplete Backup
4. Terminated Employees - Un-revoked Credentials
5. Controversial Customer Site - Poor Account Security

		Impact				
		Very Low	*Low*	*Medium*	*High*	*Very High*
Likelihood	*Very High*	🟨	🟧	🟧	🟥	🟥
	High	🟨	🟨	🟧	🟧 1	🟥
	Medium	🟩	🟨	🟨	🟧 4	🟥 2
	Low	🟩	🟨	🟨 3	🟨	🟧 5
	Very Low	🟩	🟩	🟨	🟨	🟨

Table 1: Risk Matrix

5 Appendix 2. Risk Register

Table 2: Risk Register

Threat Pairs		Current Risk				Residual Risk		
Vulnerability	**Threat Source**	**Likelihood**	**Impact**	**Risk Estimation**	**Mitigation Stratgy**	**Likelihood**	**Impact**	**Risk Estimation**
Terminated Employee	Un-patched Vulnerability	When an employee is let go often there is some resentment. With fewer than 200 employees and regular patches being made to the platform it is likely terminated employees have knowledge of platform vulnerabilities.	Dependent on the specifics of the known vulnerability, potentially high.	High	Reduce knowledge and access of exiting employees.	Reduced	Equal	Medium
Disgruntled Employee	Avoidable Logging	While logging is extensive, a technical employee is likely to know a means to disable it while acting as an attacker.	Potentially very high	High	Employee background checks, monitor job satisfaction, ensure logging is complete.	Reduced	Equal	Medium
Hardware Failure	Incomplete Backup	Regular backups are taken but in the event of (an unlikely) sudden hardware failure it is likely the latest data will still be lost.	Medium, recent data loss may be unacceptable for some customers.	Low	Run applications in new data centers in sensible locations, reduce backup delay.	Equal	Equal	Low
Terminated Employee	Un-revoked Credentials	Possible credentials of terminated employees may not be revoked immediately.	High (as above)	Medium	Enforce policy to immediately revoke access, even before termination.	Reduced	Equal	Low
Controversial Customer	Poor Account Security	Controversial customers that are known users of the platform open the potential for targeted attacks. This is less likely.	DDoS attacks still have a high impact on the platform.	Medium	Guard knowledge of company IP ranges and current customers.	Reduced	Equal	Low

6 Appendix 3. Threat Tree

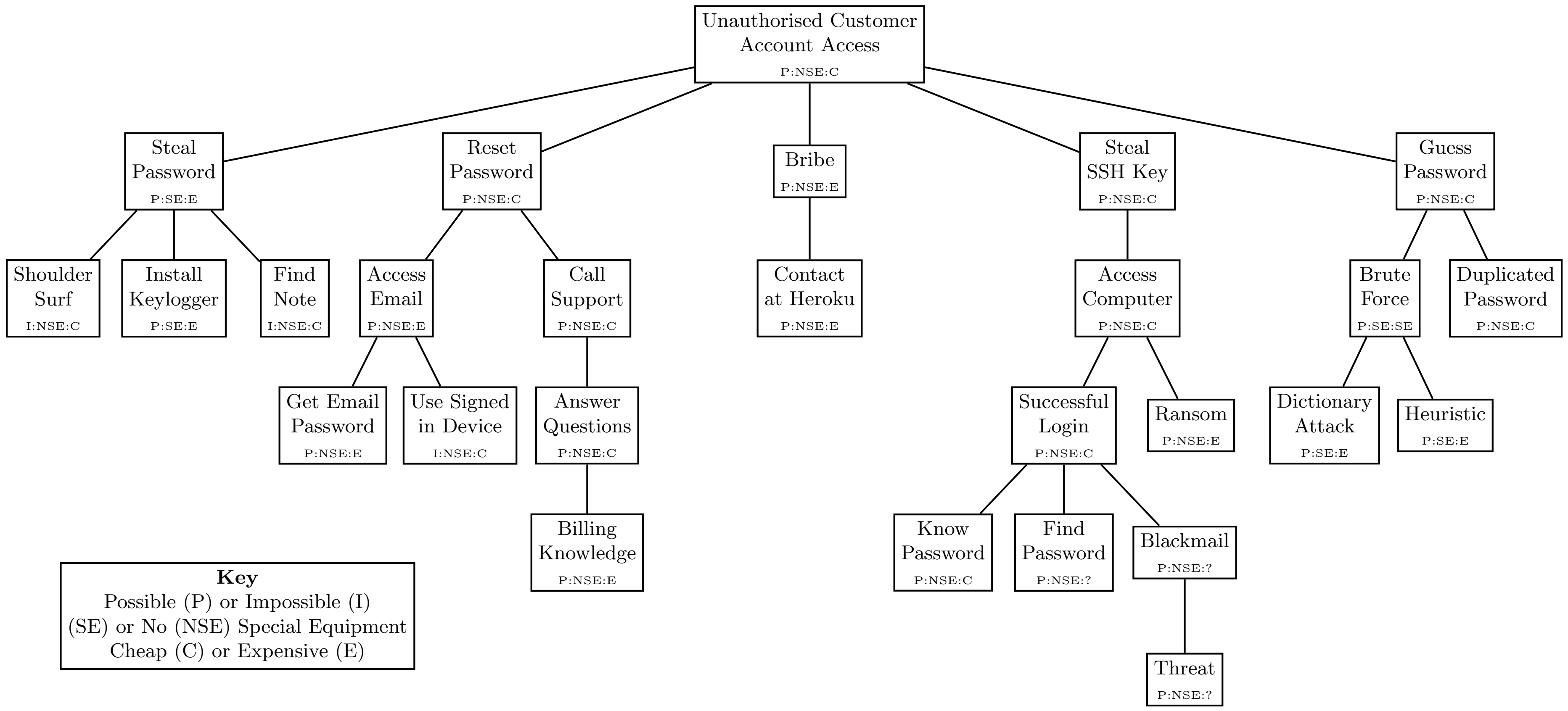
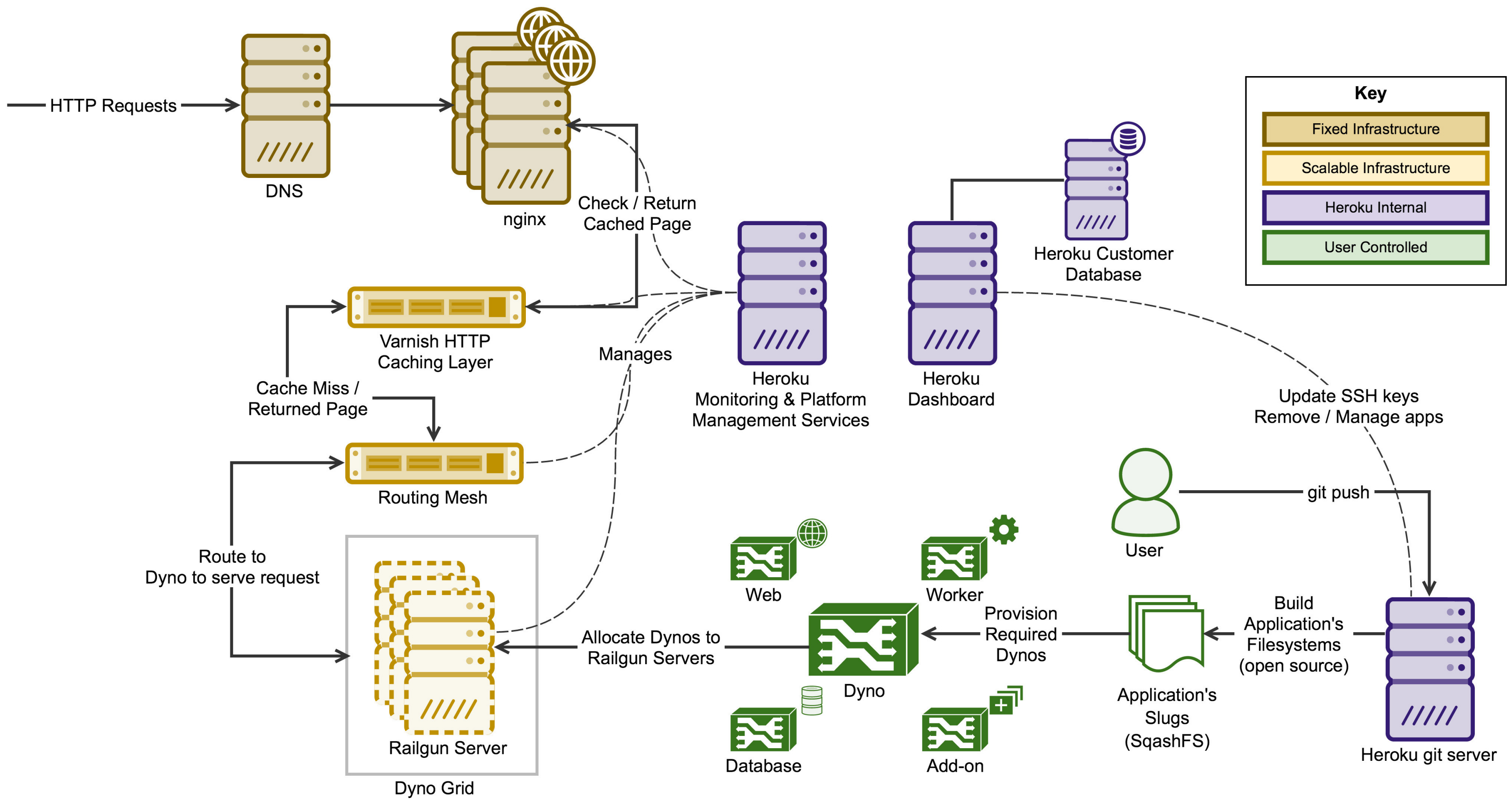


Figure 1: User Account Access Threat Tree

7 Appendix 4. System Architecture



HTTP Requests
DNS
nginx
Check / Return Cached Page
Varnish HTTP Caching Layer
Manages
Cache Miss / Returned Page
Routing Mesh
Route to Dyno to serve request
Railgun Server
Dyno Grid
Allocate Dynos to Railgun Servers
Heroku Monitoring & Platform Management Services
Heroku Dashboard
Heroku Customer Database
Key
Fixed Infrastructure
Scalable Infrastructure
Heroku Internal
User Controlled
Update SSH keys Remove / Manage apps
User
git push
Web
Worker
Dyno
Database
Add-on
Provision Required Dynos
Application's Slugs (SqashFS)
Build Application's Filesystems (open source)
Heroku git server

Figure 2: Architecture of the Heroku Platform

Bibliography

[1] Auth0. Refresh tokens. https://auth0.com/docs/refresh-token, 2015.

[2] P. Brodzinski. Top 6 problems with risk management, point 2. http://brodzinski.com/2006/11/top-6-problems-with-risk-management.html, 2006.

[3] J. Bryant. How i almost lost my $500,000 twitter username. http://hackticool.com/post/75171875746, 2014.

[4] BuiltWith. Heroku usage statistics. https://trends.builtwith.com/hosting/Heroku, 2015.

[5] Crunchbase. Heroku profile. https://www.crunchbase.com/organization/heroku#/entity, 2015.

[6] C. Egan. Heroku outage analysis. http://charlieegan3.com/blog/2015/10/15/heroku-outage-analysis.html, 2015.

[7] A. Ely. Adam ely: Summary. https://www.linkedin.com/in/adamely#summary, 2015.

[8] Heroku. Openssl heartbleed security issue, incident 606. https://status.heroku.com/incidents/606, 2014.

[9] Heroku. Two-factor authentication now generally available. https://blog.heroku.com/archives/2014/9/25/two-factor-authentication-ga, 2014.

[10] Heroku. About. https://www.heroku.com/what, 2015.

[11] Heroku. Acceptable use policy. https://www.heroku.com/policy/aup, 2015.

[12] Heroku. Adding keys to heroku. https://devcenter.heroku.com/articles/keys#adding-keys-to-heroku, 2015.

[13] Heroku. Build packs. https://devcenter.heroku.com/articles/buildpacks, 2015.

[14] Heroku. Customers. https://www.heroku.com/customers, 2015.

[15] Heroku. Ddos. https://www.heroku.com/policy/security#ddos, 2015.

[16] Heroku. Dyno types. https://devcenter.heroku.com/articles/dyno-types#setting-dyno-type, 2015.

[17] Heroku. Heroku incident log. https://status.heroku.com/past, 2015.

[18] Heroku. Heroku platform languages. https://devcenter.heroku.com/start, 2015.

[19] Heroku. Heroku uptime. https://status.heroku.com/uptime, 2015.

[20] Heroku. Http caching. https://devcenter.heroku.com/articles/http-caching, 2015.

[21] Heroku. Http routing. https://devcenter.heroku.com/articles/http-routing, 2015.

[22] Heroku. Incident 689. https://status.heroku.com/incidents/689, 2015.

[23] Heroku. Incident 773. https://status.heroku.com/incidents/773, 2015.

[24] Heroku. Network security. https://www.heroku.com/policy/security#netsec, 2015.

[25] Heroku. Pci. https://www.heroku.com/policy/security#pci, 2015.

[26] Heroku. Physical backups on heroku postgres. https://devcenter.heroku.com/articles/heroku-postgres-data-safety-and-continuous-protection#physical-backups-on-heroku-postgres, 2015.

[27] Heroku. Security researcher hall of fame. https://www.heroku.com/policy/security-hall-of-fame, 2015.

[28] Heroku. Security staff. https://www.heroku.com/policy/security#security_staff, 2015.

[29] Heroku. Slug compiler. https://devcenter.heroku.com/articles/slug-compiler, 2015.

[30] Heroku. System configuration. https://www.heroku.com/policy/security#system_configuration, 2015.

[31] Heroku. Vunerability report. https://www.heroku.com/policy/security#vuln_report, 2015.

[32] Heroku2. New heroku dashboard. https://blog.heroku.com/archives/2014/8/5/new-dashboard-and-metrics-beta#new-heroku-dashboard, 2014.

[33] Heroku2. Collaborator privileges. https://devcenter.heroku.com/articles/sharing#collaborator-privileges, 2015.

[34] Heroku2. Disater recovery: Customer applications and databases. https://www.heroku.com/policy/security#dr_customer_apps, 2015.

[35] Heroku2. Employee screening. https://www.heroku.com/policy/security#employee_screening, 2015.

[36] Heroku2. Environment safeguards. https://www.heroku.com/policy/security#environment, 2015.

[37] Heroku2. Firewalls. https://www.heroku.com/policy/security#firewalls, 2015.

[38] Heroku2. Isolation and security. https://devcenter.heroku.com/articles/dynos#isolation-and-security, 2015.

[39] Heroku2. Physical security. https://www.heroku.com/policy/security#physsec, 2015.

[40] Heroku2. Port scanning. https://www.heroku.com/policy/security#portscan, 2015.

[41] Heroku2. Privacy policy. https://www.heroku.com/policy/privacy, 2015.

[42] Heroku2. Spoofing. https://www.heroku.com/policy/security#spoofing, 2015.

[43] Heroku2. Vulnerability management. https://www.heroku.com/policy/security#vuln_management, 2015.

[44] T. Hoff. Heroku - simultaneously develop and deploy automatically scalable rails applications in the cloud. http://highscalability.com/heroku-simultaneously-develop-and-deploy-automatically-scalable-rails-applications-cloud, 2009.

[45] M. Honan. How apple and amazon security flaws led to my epic hacking. http://www.wired.com/2012/08/apple-amazon-mat-honan-hacking/, 2012.

[46] B. Lakshmi. When to use nonce? - security stack exchange. http://security.stackexchange.com/a/19834, 2012.

[47] legislation.gov. Data protection act. http://www.legislation.gov.uk/ukpga/1998/29/section/13, 1998.

[48] R. Murray-West. Facebook campaign by angry hsbc customers over new online security key. http://www.telegraph.co.uk/finance/personalfinance/bank-accounts/8725302/Facebook-campaign-by-angry-HSBC-customers-over-new-online-security-key.html, 2011.

[49] T. Rautonen. Introduction to paas and heroku, slide 11. http://www.slideshare.net/trautonen/paas-heroku, 2015.

[50] Salesforce. Heroku security, privacy and architecture. https://help.salesforce.com/servlet/servlet.FileDownload?file=015300000037zDoAAI, 2015.

[51] Splunk. Heroku selects splunk. http://www.splunk.com/view/SP-CAAAFP4, 2010.

[52] J. Stanley. Digital ocean private networking is not private. http://incoherency.co.uk/blog/stories/digital-ocean-private-network.html, 2015.

[53] R. Stiennon. Why risk management fails in it. http://www.networkworld.com/article/2160724/tech-primers/why-risk-management-fails-in-it.html, 2012.