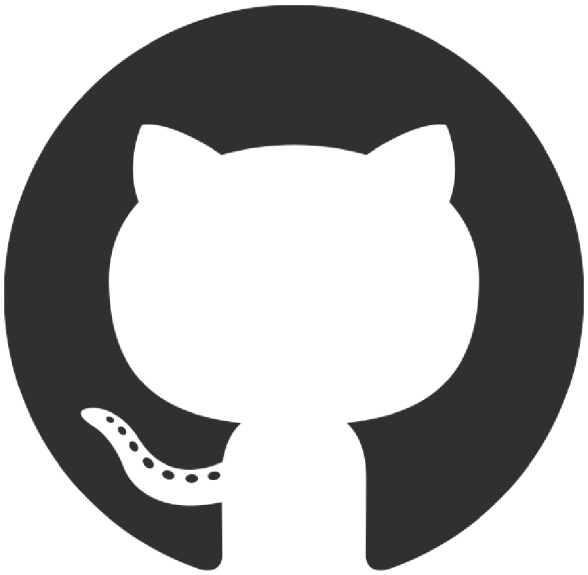


GitHub.com Services Continuity and Incident Management Plan

Change Record

Date	Version	Change Comments
6.25.2020	1.2	Initial customer release of document
9.21.2021	1.3	Updated document to reflect internal service tiers, incident management processes, changes to infrastructure diagrams.

V.1.3 / September 2021

GitHub.com Services Continuity and Incident Management Plan **1**

Change Record **2**

Introduction **4**

Objectives 5

Scope 5

Impact Analysis **6**

Data Centers **8**

Service Catalog **9**

Observability **10**

Service Response & Continuity **11**

Incident Response Framework 11

Incident Commander responsibilities: 12

Communication Lead responsibilities: 12

Subject Matter Expert responsibilities 13

Incident Response Workflow 13

Recovery Response Workflow 16

GitHub Security Incident Management Process **20**

Security Incident Response 20

Vulnerability Assessment 22

Severity definitions 22

Introduction

At GitHub, we consider security, quality, and service reliability at every phase of development and operations. We continually invest in our culture of service quality excellence and transparency. The following document is intended to give you insight into our current Service Continuity capabilities and Incident Management plan, and to share GitHub's focus on improvements to the reliability of continuous operations in the production environment.

GitHub operates a highly-available system of data centers with continuous and delayed replication, ensuring a high level of service and resiliency in our operations. As our services and the supporting technologies continue to advance and evolve, we will continue to improve our processes, plans, and technical capabilities. We will review this document on an annual basis, or more frequently as needed, in order to reflect significant changes.

As part of our commitment to transparency and service reliability, we publish a monthly GitHub Availability Report, where we share root cause analyses and the resulting engineering repair plans for any incidents. We also offer our enterprise customers assurance through our GitHub Online Services SLA for service levels which we guarantee.

Sincerely,

Michael Hanley, Chief Security Officer

Objectives

To provide interested external parties a high-level summary information about our services and their resiliency.

At GitHub, we recognize that the unexpected can and does occur - from simple situations to major outages. This document summarizes the measures GitHub has taken to respond to major or significant business disruptions. Here you will find an overview of our program, our methodology, how we work, and our focus on delivering what our customers need:

- To immediately mobilize a core group of leaders to assess the situation and quickly deploy personnel to restore or move the services to an alternate site as necessary.
- To effectively meet the business requirements at the alternate site during the recovery period.
- To restore operations at the alternate site within the recovery period to minimize the impact on the critical business processes.
- If appropriate, to stage the restoration of operations to the original site to resume full processing capabilities.

Scope

The scope of this document addresses our response to and recovery from major disruptions to GitHub (github.com) services. Documentation for GHAE, GHES, employee safety, and emergency response (such as Enterprise Crisis Management Plans) is outside this document's scope.

This document includes:

- Impact Analysis & Prioritization Overview
- Response Overview
    - Incident Management Process
    - Recovery Process
    - Security Incident Management Process

This document does not include the following:

- Procedures for life safety emergency response
- Pandemic response plan
- Executive procedures for crisis communications and management

Impact Analysis

We consider the criticality of our services by looking at four categories:

- **Operational Scope (OS)**
  - The breadth of an event .
- **Trust / Reputational**
  - Stakeholders = Customers, Partners, and / or Shareholders.
  - Threat to customer perception and trust in our services or brand; failure to meet SLA whether that is for partners or commercial customers.
- **Legal, Compliance, and Environmental (LCE)**
  - Consequences of a risk resulting in restrictions or fines to a specific service or the entire company.
- **Enterprise Value**
  - Financial risk to the company.

Rating	OS	Trust	LCE	EV
Critical	Enterprise-wide impact to business operations globally	Permanent loss of trust; Failure to meet SLA for all users	Global restrictions in conducting business on certain product lines, markets, or geographies	Material reduction is market capitalization, significant draw of liquidity reserve. OR financial impact of >1.5B
High	Significant impact to business operations within one or more businesses / geographies	Significant loss of trust; Failure to meet SLA for a large set of users	Prohibited from conducting business on certain product lines, markets, or geographies	Significant reduction is market capitalization, substantial draw of liquidity reserve. OR financial impact of >300M
Moderate	Moderate impact to businesses or geos. Major inability to deliver on business objectives	Moderate loss of trust; Failure to meet SLA for a moderate set of users	Moderate fines or limitations on conducting business on certain product lines, markets, or geographies	Moderate reduction is market capitalization, limited draw on operating cash flow. OR financial impact of >10-75M
Minimal	Minimal impact within one business or geo	Limited loss of trust; Failure to meet SLA for a small set of users	Limited to minimal action against the company, with limited effects on operations	Minimal reduction is market capitalization, limited draw on operating cash flow. OR financial impact of <10M
No Impact	No material impact	No material impact	No material impact	No material impact

Prioritization

GitHub Senior Leadership provides guidance for setting our response and recovery goals for services. Based on that guidance, efforts prioritize the response and recovery of the most essential services first. These are internally reflected within our Service Catalog as tiers and externally reflected in our GitHub Online Services SLA .

Tiers are assigned to services in our Service Catalog to describe their importance to business continuity and the order in which they should be fixed when incidents occur. Tiers are numbered 0 - 3 where Tier 0's uptime is critical to business and should be first to fix, and Tier 2's uptime is less important. GitHub Service Tiers are defined as follows.

Tier	**Criteria**	**Examples** (not exhaustive)
Tier 0 - Dialtone services	• Infrastructure services requiring dialtone reliability. • Services that are foundational to GitHub's uptime-guaranteed services. • Services that are required for fixing all other failures. Customers expect us to prioritize fixing these first. •	• DNS • Storage • Identity • Global Load Balancer (GLB) • Cloud and data center infrastructure
Tier 1 - Critical services	• The default tier for products and services in GA. • Services that customers rely on to get work done. • Services with an external SLA (uptime-guaranteed services). • Services enabling critical communication and the ability to recover from any failure. • GitHub mobile apps. • First or third-party dependencies of services in this tier.	• Pages • Issues • Webhooks • GitHub for mobile • Actions • Observability incident response tools
Tier 2 - Non-critical services	• Informational customer services. • Non-essential services that add value to the customer experience. • Services in incubation that are in production or deployed, but not publicly available. • GitHub command line interfaces (CLIs). • Deprecated services that are publicly available. • Internal tools or third-party services for business operations. • Special-purpose services not useful to general customer audiences. • First or third-party dependencies of services in this tier.	• Internal support tools• GitHub CLI • GitHub Stars • Marketing pages • Product documentation
Tier 3 - Zero-impact services	• Experimental projects. • Services without SLOs. • Non-production or pre-production services that are not deployed.	• Hackathon projects • Personal and team experiments

Data Centers

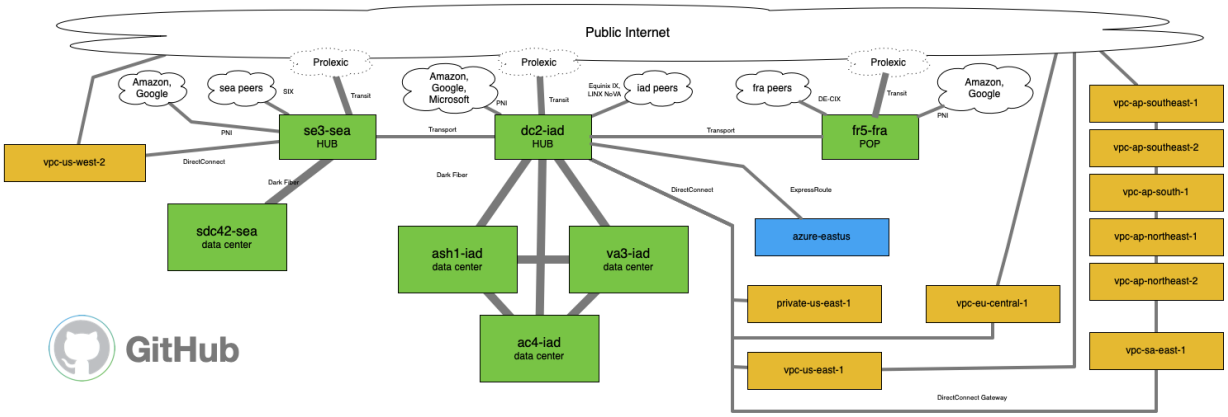
GitHub.com operates in a high availability environment built on a heterogeneous mix of data center and network suppliers. It employs a multisite strategy within a primary region across 3 data centers.

GitHub has redundant production network edges in Ashburn, VA and Seattle, WA that are solely stateless transit sites with backhaul to our data centers. Git data is replicated between Sterling, Reston, and Ashburn, VA to provide data center redundancy. GitHub houses encrypted off-site backups of repository data using AWS S3 object storage in US East 1 (N. Virginia) and US West 2 (Oregon).

GitHub.com services and data are primarily hosted in Virginia (USA) out of three data centers configured in an active-active-active configuration built with sufficient capacity to provide uninterrupted service with two out of three sites operating.

Our data center providers are as follow:

- QTS located in Sterling, VA
- Coresite located in Reston, VA
- Sabey located in Tukwila, WA and Ashburn, VA
- Equinix located in Ashburn, VA and Seattle, WA
- AWS EC2 and related network offerings in ~a dozen regions globally
- AWS S3 in the US East (us-east-1, N. Virginia) and US West 2 (us-west-2, Oregon) regions
- Azure



The image above shows our MySQL database configuration across the three Virginia data centers.

Customers can also create backups of their data, GitHub provides information on how customers may backup their own data as a safeguard (see this GitHub Help article for further context and instructions: https//help.github.com/en/articles/backing-up-a-repository.

Service Catalog

GitHub's Service Catalog is an internal website/product and API that lists all internal services and serves as a catalog and policy enforcement tool. This project incorporates a concept called "durable ownership", itself an internal project within GitHub. This durable ownership framework establishes expectations and policies around GitHub's internal and external products, services, and features in production. Durable ownership relies on a collection of roles (team, volunteers, business sponsor, product sponsor) and a set of escalation policies to maintain ownership continuity.

The Service Catalog also provides valuable cross-functional visibility and discovery capabilities that improve cross-team cooperation during incidents. It acts as a standardized interface for teams, e.g., communication channels, service dependencies, SLOs, and support level for the service. This service establishes clear expectations, limits, and performance expectations across teams. These performance expectations are weighted checklists written by Subject Matter Experts on non-business logic requirements for production services. Tiering of our services based on their response and recovery priority is reflected internally within the Service Catalog.

Ownership (using durable ownership) declared in the service catalog is required to deploy applications to our production environments. Each service is required to fill out a durable ownership configuration known as `ownership.yaml`. These configuration files include information (located within the @github organization) define crucial pieces of ownership for the service:

- **Team:** The owner of the app's availability, representing organizational buy-in to the product/service/feature.
- **Communication channels:** How to reach the team for a variety of different scenarios (during incidents, interrupts, migration work, vulnerability management) that typically include an @mention, slack channel, and repository for issues.
- **Repository:** Project repositories within the @github organization that represent the intellectual property for this service.
- **Service Dependencies:** other entries in the Catalog that this service depends on.
- **Service level:** an enumeration of possible levels of support that `Team` provides.
- **Organizational sponsorship**: individuals in product and the executive team that sponsor this service.
- **SLOs:** user expectations for the service that the `Team` provides.

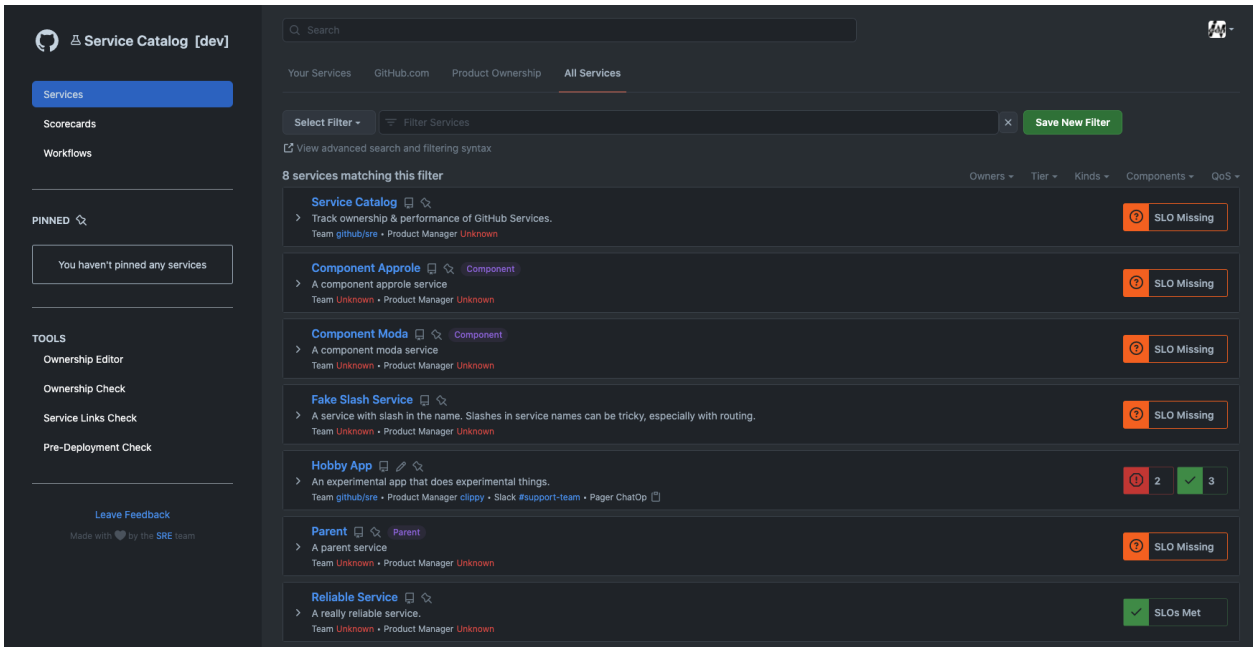
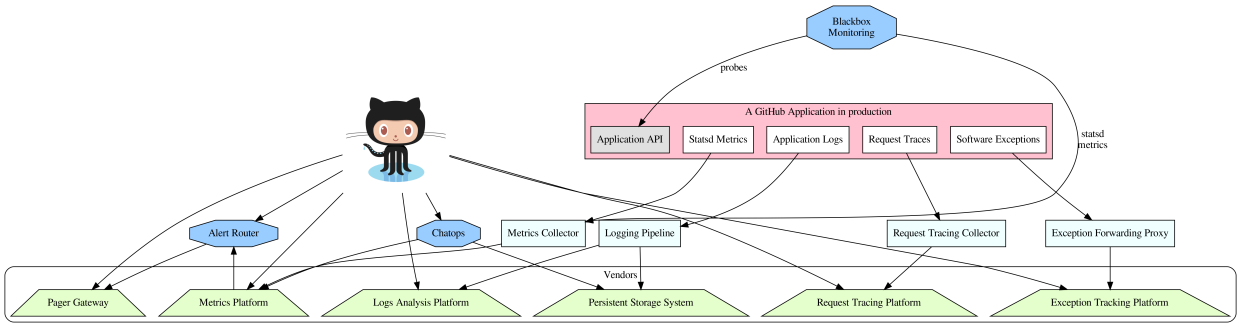


Figure: Example rendering of the Service Catalog

Observability

Our observability stack comprises a mix of applications hosted by us (in our data centers, AWS, and Azure) as well as external vendors' solutions.



Plain text logs can be visualized, aggregated and summarized in tools like Splunk. However, the most recent logs can also be accessed on the individual host's command line in case the logging pipeline runs into issues. A log rotator rotates the logs on the hosts, so only the most recent data is available.

Metrics cannot easily be consumed on the command line; however, metrics consumption on the command line also often doesn't provide additional information compared to the log files.

Service Response & Continuity

The following outlines our processes for incident response and recovery.

Incident Response Framework

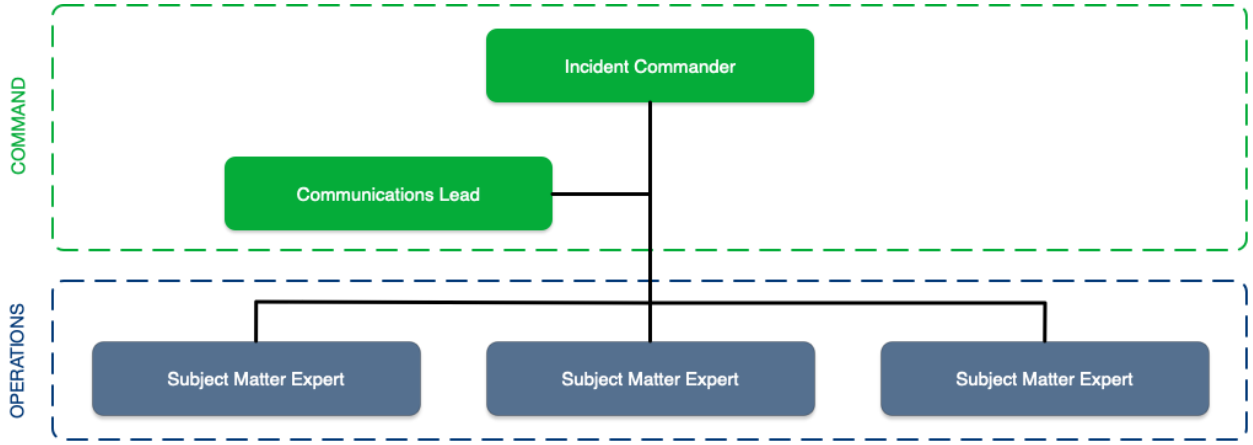
Our customers take on a critical dependency relying upon us and our services. Our customers need to be able to trust and rely on the people running the service. This framework defines the roles and workflow for managing an Incident to build trust and increase transparency and credibility with our customers. Security incidents are handled outside of this response framework via GitHub's security incident management process, which is outlined later in this document.

Roles

There are three broad roles for our incident response teams at GitHub. We operate within an expandable response structure where certain roles only have one person per incident (e.g. Incident Commander), whereas other roles can have multiple people if the incident warrants (e.g. Subject Matter Expert).

1. **Incident Commander**
2. **Communications Lead**
3. **Subject Matter Expert**

Overview



Incident Commander

Each service listed on https://githubstatus.com has a dedicated on-call rotation whose members are trained as Incident Commanders. Each rotation has primary and secondary on-callers. These on-callers are expected to be Subject Matter Experts in their respective services.

For incidents impacting a single service, the owning team is responsible for managing them. When paged, the primary on-caller will assess if they can run the incident on their own (i.e. serving as Incident Commander and Subject Matter Expert). If the incident is complex, they will

page their secondary to assume the Subject Matter Expert role so that they can focus exclusively on coordination and communication tasks.

For incidents impacting multiple services, the first team to respond defaults to holding the Incident Commander role. Since the incident involves multiple services, a dedicated Incident Commander may be required. In this case, the on-caller will page their secondary to assume the Subject Matter Expert role so they can focus on the Incident Commander role.

Incident Commander responsibilities:

- Determine user impact (by consulting with each service's Subject Matter Expert)
- Act as the Communication Lead or involve a dedicated Communication Lead, if needed.
- Collect information from team members for their service's status.
- Collect proposed repair actions, then recommend repair actions to be taken.
- Delegate repair actions
- Be the single authority on system status (or delegate this to the Communication Lead)

Communication Lead

In most cases, the Incident Commander performs the duties of the Communication Lead. If an incident is complex, the Incident Commander may designate another person to hold the Communication Lead role.

The communication lead drives internal and external communication efforts using information from the Incident Commander. They understand technical updates, conflate them with business impact, and convert them into status updates for both internal and external customer consumption.

During complex incidents, it's not always possible for the Incident Commander to also focus on internal and external communication. This is where the Communication Lead comes in. When a complex incident is declared, the Communication Lead will take over internal and external communication allowing the Incident Commander to focus on incident mitigation. It's up to the Incident Commander to decide when a Communication Lead needs to be involved.

Communication Lead responsibilities:

- Update the external status page
- Update internal incident command channel with an incident summary
- Liaise between the incident commander, executives, marketing, legal, and PR.

Subject Matter Expert

A Subject Matter Expert is a domain expert or designated owner of a component or service that is part of the GitHub technology stack. If an incident is complex, the incident commander and communications lead cannot be expected to know the internal workings of each service and component. Therefore, the Incident Commander can bring a Subject Matter Expert for that service to help identify and fix issues. The Incident Commander will rely on this Subject Matter Expert to communicate options and conduct mitigating actions. The Service Catalog is used to identify Subject Matter Experts during an incident.

Subject Matter Expert responsibilities

- Determine user impact (using SLOs, dashboards, playbooks).
- Determine potential causes and work on a plan to mitigate user impact.
- Identify problems with a service's dependencies, and escalate to the responsible team.
- Provide updates to the Incident Commander, specifically for CAN reports:
    - Condition: What is the current state of the service? Is it healthy or not?
    - Actions: What actions need to be taken if the service is not in a healthy state?
    - Needs: What support does the Subject Matter Expert need to perform an action?
- Once a fix is deployed, validate user impact is resolved

Incident Response Workflow

The Incident response workflow starts when our telemetry has detected degradation of service, an SLO breach is reached, or through manual communication.

1. An oncaller receives notification that their service may be experiencing issues. If the oncaller confirms their service is in a degraded state in a way that impacts their users, they head to `#incident-command` and update the service's status color to `yellow` (degraded or unconfirmed) or `red` (interruption of service), depending on the severity of the issue.
2. A degradation to `red` or `yellow` triggers an escalation to other internal stakeholders.
3. These responders assess the severity of the issue. If necessary, they declare a public incident and coordinate efforts in a recorded video conference and shared documents. An issue for the incident is created via automation in a specialized repo to track the work.
4. Incident managers provide regular internal and external status updates until the issue is resolved.
5. When we are confident that the underlying issue is addressed and functionality restored, we resolve the incident and transition the affected services back to `green` (normal operation).

Classification

The first step in any incident response process is to determine what constitutes an incident. Incidents are then classified by severity using "sev" definitions, with lower numbered severities being more urgent. Operational issues are classified at one of these severity levels. Anything above a Sev-2 is automatically considered a "major incident" and gets a more intensive response than a normal incident.

If we are unsure which level an incident is (e.g., not sure if Sev-2 or Sev-1), we treat it as the higher one. During an incident is not the time to discuss or litigate severities.

Sev-0 (Disaster)	**Critical issue that warrants public statement and liaison with executive teams.** ● One or more systems in a critical state globally due to a physical, geopolitical situation (DC & network capacity, workspace, infrastructure) or a security issue that results in GitHub not being delivered at a level that is intended. ● Irrecoverable data loss or data corruption due to platform defect or security incident. ● Customer-data-exposing security vulnerability has come to our attention.	Page an Incident Manager in Slack or escalate within the current Incident Responder team.
Sev-1 (Single/multi-service/region impact)	**Issue that warrants public notification.** ● A system is in a degraded or disrupted state and is actively impacting a subset of customers. ● Impact is significant: either widespread or sustained (> 4h). ● Functionality has been impaired, potentially breaking SLA.	Service owner team to investigate and mitigate.
Sev-2	**Issue that may have customer impact if not mitigated in the near future.** ● A system showing a trend towards an unhealthy state such as disk space usage reaching a warning level.	Service owner team to investigate and mitigate.
Sev-3	**Operational issue** ● A single transient failure of a cron job which is due to run again.	Service owner team to investigate and mitigate.

Status Colors

- `green` means everything is operating as expected.
- `yellow` means a service is experiencing degraded performance but the service remains usable. For example, this could mean delays in email notifications, slow loading of webpages, or GitHub Pages builds taking too long.
- `red` means a service or product is unusable. This could be a very high rate of 500s for the website, GitHub Pages failing every build in the past hour, or a high ratio of git operations failing.

Incident responders use bespoke chatops to create, update, and resolve incidents. We constrain public communications to be consistent with the GitHub voice during high stress incidents. The following table highlights common terminology we use:

Status	Update descriptions
Investigating	**General** ● We are investigating reports of elevated error rates. ● We are experiencing a small increase in exception rate and are investigating. ● We are investigating reports of service unavailability. **Web** ● We are investigating increased error rates on GitHub.com. ● We are experiencing elevated error rates for user dashboards and are investigating. ● We are investigating page timeouts and delays to user timeline updates. **Notifications** ● We are investigating delays in notification delivery. ● We are investigating delays in email delivery.
Identified Work is still being done	We've identified the cause of the incident. We are working on a fix.
Identified Fix is being released	**Example descriptions** ● Users may experience delay in timeline updates and searches. ● Notification delivery times are back to normal. Updates to user timelines are currently delayed. ● We are continuing to monitor the recovery of timeline delays. ● We have identified the source of connectivity issues to GitHub.com and continue to monitor as the site recovers. ● Email delivery is running behind, you may see delays in receiving notifications. ● Updates to user timelines are currently delayed, recent activity may be missing. ● We have addressed the source of the exceptions and are continuing to monitor as the site recovers. ● Notification delivery and updates are running behind. You may see delays in email and web notifications as well as unsubscribes. ● The backlog of notifications is being worked through. We will post the next update once all notifications have been delivered.

In the case our chatops is down, an incident manager can manually control both Twitter and GitHub's External Status page (https://www.githubstatus.com/).

Manually controlling githubstatus.com

If an incident manager needs to manually create, delete, or edit resources on githubstatus.com, they will follow internal procedures to log into the service directly via our single sign-on portal.

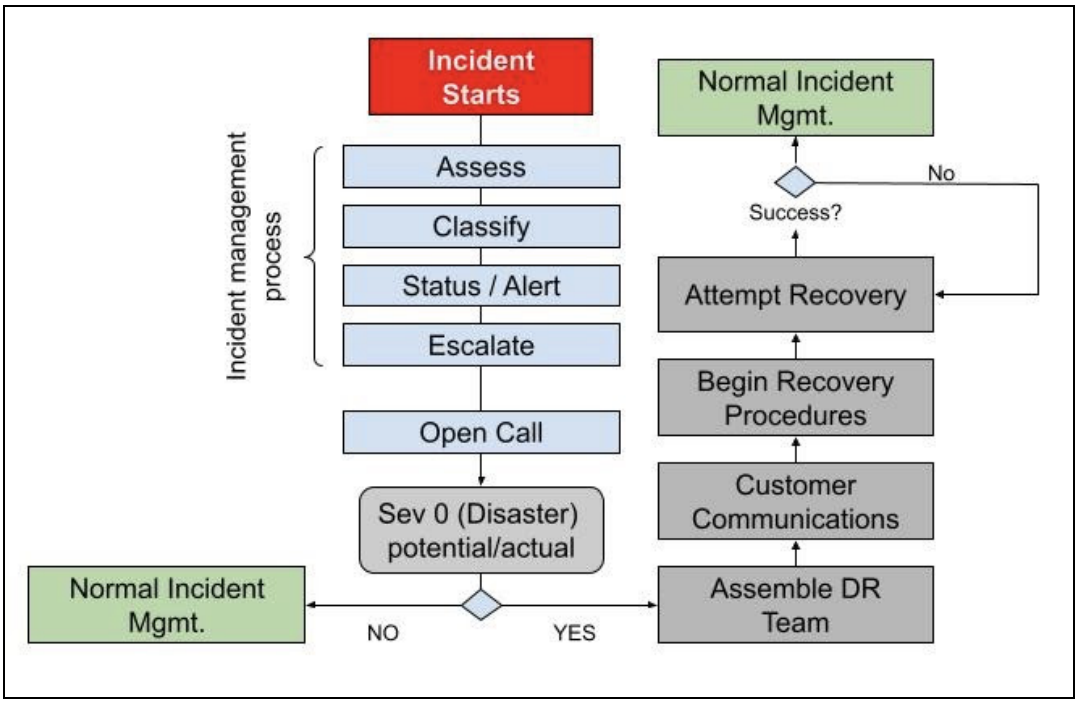
After the Incident
After the incident, incident responders follow through with more detailed analysis and long-term fixes by completing a standardized issue PIR (post incident response) template. When the responders are ready, they indicate through the issue that they are ready for review. Weekly reviews discuss these issues and investigate common trends and push for specificity, resolution, and accountability.

After significant issues, product teams will create a public blog post to provide additional transparency.

On the first Wednesday of each month, we'll publish a GitHub Availability Report, including a description of any incidents that may have occurred and provide an update on how we are evolving our engineering systems and practices in response.

Recovery Response Workflow

If the incident is classified as a Sev-0 incident and cannot be mitigated in the primary site, the corresponding workflow *expanding* our response process can be activated. The following is a high level flow diagram of our workflow.



Disaster Scenarios

The following are scenarios that may trigger a recovery-focused response.

Disaster Scenarios	Activate if
Technology disruption	Hardware or software mitigations are not effective, and there is no clear line of sight of recovery time.
Broad Regional Event (Virginia) - Includes regional network and power outage	All three of our availability zones are down with no clear line of sight of mitigation or recovery time.
Data center Workforce Disruption	Workforce is disrupted to the point that hardware maintenance cannot be initiated nor guaranteed to complete.
Data loss scenario	We determine that shutting down the service is the best mitigation to avoid further data loss scenarios.
Cybersecurity attack	We determine that shutting down the service is the best mitigation to avoid damage from the attack.

Recovery checklist

Below is an example checklist outlining the steps a recovery team would follow to execute recovery for an unresolved Sev-0 incident:

Action	Notes
Initial Assessment & Classification	Assumption: Completed via incident response
Open internal communications bridge	Assumption: Video conference bridge open
Activate upon inability to resolve Sev-0 incident	If activation is not required, stand down and close the bridge.
Escalate	Escalation to SVP and VPs to assemble Recovery team
Assemble Initial Recovery team	The recovery team is composed of: - SVP of Engineering - Technology and Product area VP - Appropriate service and infrastructure engineers in charge of the disaster recovery
Open bridge	Open a new one or reuse incident bridge as appropriate

Initiate customer communication procedures	
Conduct detailed assessment with recovery team	
Review recovery priorities	Tier 1 services should be recovered first, then Tier 2
Initiate recovery procedures	Allocate team member resources for the recovery activities
Communicate as necessary during recovery	
Verify recovery and communicate	
Stand down and close the bridge	

Phases & Roles

Response Phase	Authority to declare a disaster
	Recovery Teams & Roles
	Team Notification Process
Recovery Phase	Service Recovery steps as documented per service
Verification Phase	Verification steps post recovery
Closing Phase	Disaster Closure actions

Response Phase

At GitHub the following individuals have the authority to declare a disaster:

Group	**Title**
Infrastructure	Sr. Director or Vice President
Engineering	SVP Engineering or Designee
Security	Chief Security Officer
CEO Office	Chief Executive Officer or Designee

Recovery Roles

Recovery members and their roles:

Role	Description
Communications Lead	Supports site incidents and post incidents communications both internally and externally.
Support Lead	Supports customer escalations due to site incidents and recovery operations.
Operations Lead	Coordinates all of the work needed to execute a recovery. Inclusive of impacted services and infrastructure.
Finance Lead	Supports credit calculations and issuance for SLA violations due to the incident.
Legal Affairs Lead	Supports and provides legal advice and guidance on/to the recovery v-team.

Recovery, Verification Phase

Recovery and verification steps overview for services.

1. Location of production deployments
2. Service and platform dependencies should be noted along with information of how important they are to recovery.
3. The recovery procedures must be detailed to the level that any assigned oncall responder can recover the service.
4. The recovery procedures must include steps for complete environment failover. I.e. from primary site to secondary site.
5. Any steps warranting other engineering team involvement should be noted along with contact information.
6. It is recommended to have all DR scripts with comments to be present
7. Any steps that can be executed in parallel should be folded in as sub steps of a main step.
8. All steps to be noted in a markdown file explicitly with no references to GitHub resources (wikis, repos, etc.)
9. Detailed recovery steps with pre & post validations called out explicitly.
10. Clear "smoke/ functional test” verification steps noted with expected result.

Closing Phase

The closing phase applies to services after a recovery has been performed. At a high level it includes:

- Validation that all essential systems are operational,
- Identifying items that require further attention,
- Identification of gaps,

- Creation of a document summarizing the event and "Lessons learned", and
- Participation in the Post Incident Response (PIR) review process.

GitHub Security Incident Management Process

The security of our code is critical for the continued success, growth, and viability of GitHub as a company. As such, GitHub has created a Security Policy reinforcing GitHub's transparency and accountability to our customers, users, and employees.

The GitHub Security department, under the direction of the Chief Security Officer, is responsible for the formulation and maintenance of this policy, and for responding to questions posed regarding this policy. Its program principles are as follows:

1. Ensure the confidentiality, integrity, availability, and privacy (a.k.a. the security) of all corporate and customer data handled by the company.
2. Ensure the strategic initiatives of GitHub are executed in a security aware manner with secure design in systems, applications, and processes.
3. Identify, minimize, and mitigate security-related risk and associated financial risk to the company, our partners, and our customers by executing from an enterprise-wide, holistic perspective.

Security Incident Response

GitHub is committed to maintaining the confidentiality, integrity, and availability of both our platform and the intellectual property and personal information of our users, customers, and employees. In order to ensure any compromise of these principles is understood, remediated, and disclosed to affected parties as quickly as possible, GitHub maintains a robust incident response and data breach notification capability.

The What

The GitHub Detection and Response Program provides formal, comprehensively documented procedures designed to ensure that security incidents and data breaches are handled and recorded in a consistent and complete fashion.

The GitHub Security Incident Response Plan outlines the purpose, scope, roles, responsibilities, SLAs, procedural flow, and outputs of GitHub's incident response procedures.

The GitHub Data Breach Notification Plan defines the purpose, scope, roles, responsibilities, SLAs, process flow, and outputs of GitHub's data breach notification procedures. GitHub will notify customers or employees without undue delay upon GitHub becoming aware of a Data Breach affecting customers' or employees' Protected Data, providing customers with sufficient information to allow them to meet any legal and contractual obligations in relation to breach management and onward notification and providing customers and employees sufficient information to allow them to protect themselves where possible.

GitHub will notify statutory authorities on a timely basis, including the California Attorney General and the Netherlands Data Protection Authority, in the event of a Data Breach requiring such notification.

GitHub will comply with state, federal, and international laws regarding breach notification, where there are such laws.

The Who

GitHub Corporate Information Security (CSIRT) and Product Security Engineering Response (PSIRT) are jointly accountable for the successful execution, maintenance, and documentation of all incident response procedures as well as the execution of data breach notification procedures.

GitHub's Legal Department, in partnership with the incident response functions, is accountable for maintenance and documentation of data breach notification procedures.

The incident response functions coordinate with a broad range of partners and stakeholders throughout the organization to successfully respond to security incidents. These stakeholders are defined in detail in the GitHub Security Incident Response Plan.

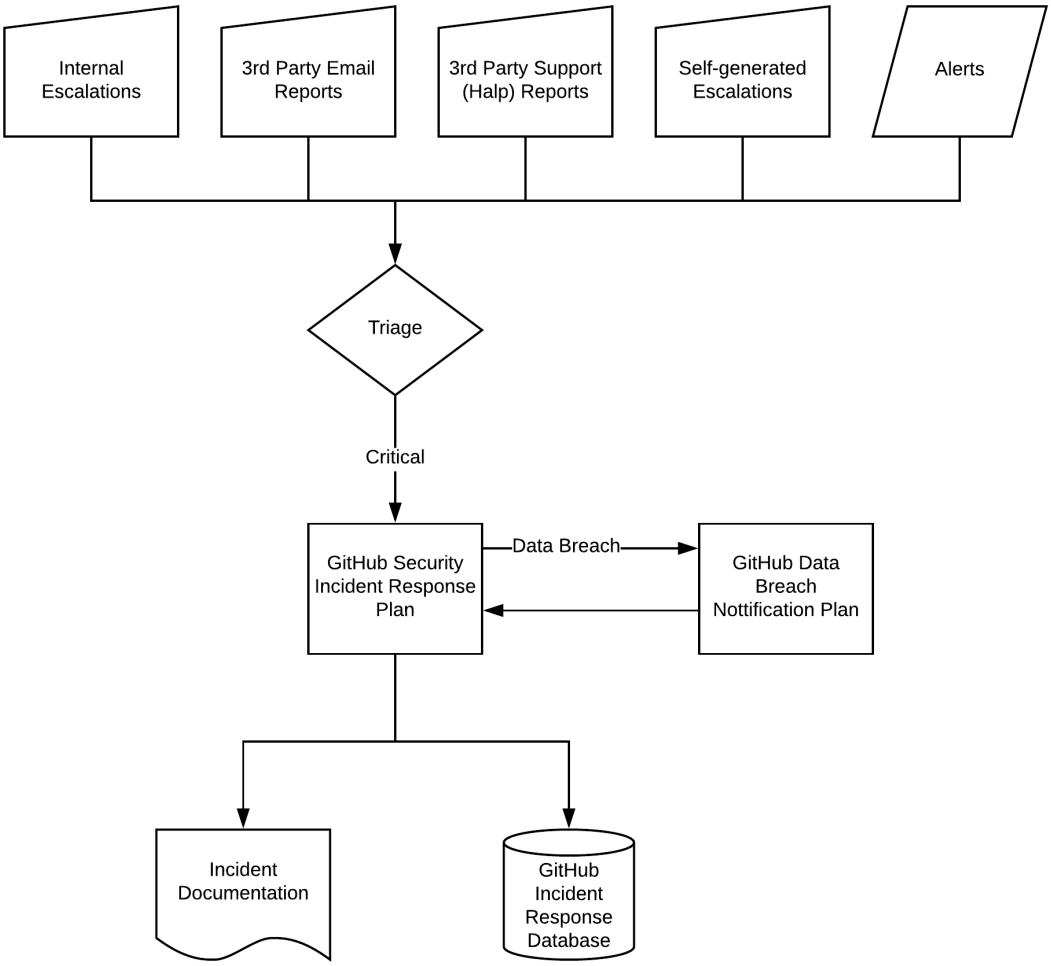
GitHub Security leadership is responsible for ensuring the incident response capability is appropriate in scope and authority and that it is staffed and funded sufficiently to ensure effective operation. In addition, leadership provides ongoing oversight and any additional support required to successfully execute these functions.

The How

Security events surfaced via GitHub's Security Detection and Response program or via internal or external escalation are triaged on a per-event basis. Those events determined to be critical are escalated as leads which are investigated as outlined by the GitHub Security Incident Response Plan. Those events resulting in a factual and significant compromise of confidentiality, integrity, or availability will be handled as formal security incidents.

Those incidents determined to be data breaches require a special set of procedures outlined in the GitHub Data Breach Notification Plan. Once those procedures are completed, the remainder of the Incident Response Plan is executed.

All security incidents complete a full lifecycle which includes a feedback and post-incident analysis process, as well as metadata recording for historical incident response metric generation.



Vulnerability Assessment

After receiving a security vulnerability, our incident response teams review and validate the impact of the vulnerability to GitHub. The incident response teams will then triage the vulnerability to partner teams across GitHub that will complete the remediation tasks. As part of this triage, the incident response teams will identify the vulnerability's severity and set a corresponding SLA to ensure mitigation efforts are properly prioritized.

Severity definitions

Critical

Critical severity issues present a direct and immediate risk to a broad array of our users or to GitHub itself. They often affect relatively low-level/foundational components in one of our application stacks or infrastructure. For example:

- arbitrary code/command execution on a GitHub server in our production network.
- arbitrary SQL queries on the GitHub production database.
- bypassing the GitHub login process, either password or 2FA.
- access to sensitive production user data or access to internal production systems.
- accessing another user's data in the GitHub Actions service.

High
High severity issues allow an attacker to read or modify highly sensitive data that they are not authorized to access. They are generally more narrow in scope than critical issues, though they may still grant an attacker extensive access. For example:

- injecting attacker controlled content into GitHub.com (XSS) which bypasses CSP.
- bypassing authorization logic to grant a repository collaborator more access than intended.
- discovering sensitive user or GitHub data in a publicly exposed resource, such as an S3 bucket.
- gaining access to a non-critical resource that only GitHub employees should be able to reach.
- using the GitHub Actions repo-scoped GitHub token to access high-risk private content outside of that repository.
- code execution in a desktop app that requires no user interaction.

Medium
Medium severity issues allow an attacker to read or modify limited amounts of data that they are not authorized to access. They generally grant access to less sensitive information than high severity issues. For example:

- disclosing the title of issues in private repositories which should be inaccessible.
- injecting attacker controlled content into GitHub.com (XSS) but not bypassing CSP or executing sensitive actions with another user's session.
- bypassing CSRF validation for low risk actions, such as starring a repository or unsubscribing from a mailing list.

Low
Low severity issues allow an attacker to access extremely limited amounts of data. They may violate an expectation for how something is intended to work, but it allows nearly no escalation of privilege or ability to trigger unintended behavior by an attacker. For example:

- signing up arbitrary users for access to an "early access feature" without their consent.
- creating an issue comment that bypasses our image proxying filter by providing a malformed URL.
- bypassing community-and-safety features such as locked conversations.
- bypassing billing & plan restrictions to gain access to paid features.
- triggering verbose or debug error pages without proof of exploitability o.r obtaining sensitive information.
- triggering application exceptions that could affect many GitHub users.
- injecting JavaScript event handlers into links, etc, which are mitigated by CSP on GitHub.com

Bounty program

https://bounty.github.com/

Software security researchers are increasingly engaging with Internet companies to hunt down vulnerabilities. Our bounty program gives a tip of the hat to these researchers and provides rewards of $30,000 or more for critical vulnerabilities.