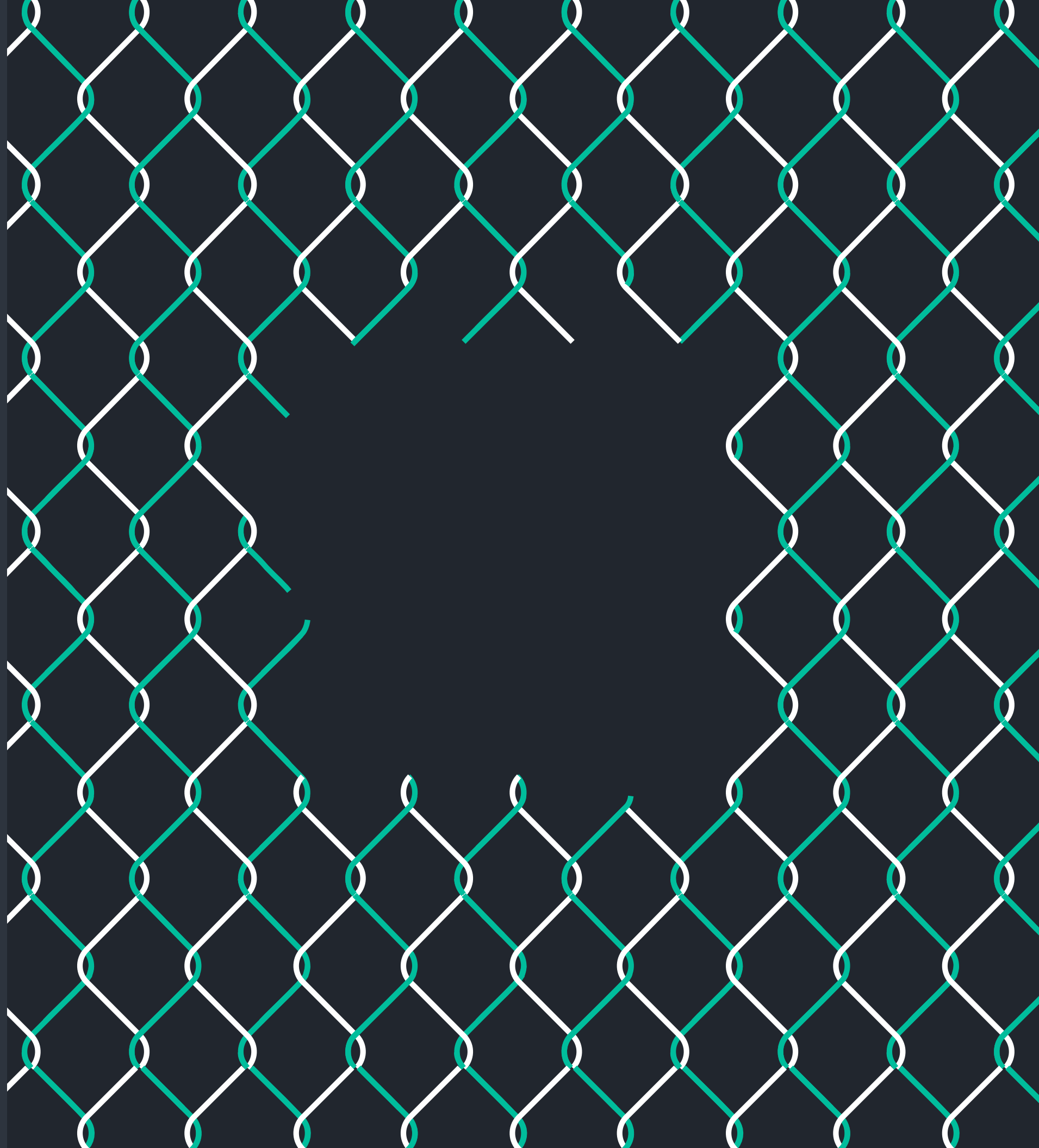




API Monitors: A False Sense of Security



Weak API Monitors Are Failing You

According to Gartner, 95% of API errors and breaches are due to human error. Conventional approaches to API testing and monitoring do not do a good job of capturing human error. For instance, a ping or contract test would not detect a problem with API specs that have not been properly written or updated. Yet most banks and enterprises handling oceans of private customer data continue to relegate API testing to conventional approaches including contract testing.

API flaws contribute significantly to the rising costs of poor quality software, which is soon to pass \$3 trillion. Conventional API testing and monitoring is not enough, and APIs are becoming the new preferred target of hackers. One API breach can expose the private information, like social security numbers, of millions of customers.

While more is being written about API security breaches, not enough is written about how to stop them. Human error is everywhere in API design and building, and can lead to functional issues that are overwhelmingly the door by which API hackers walk through.

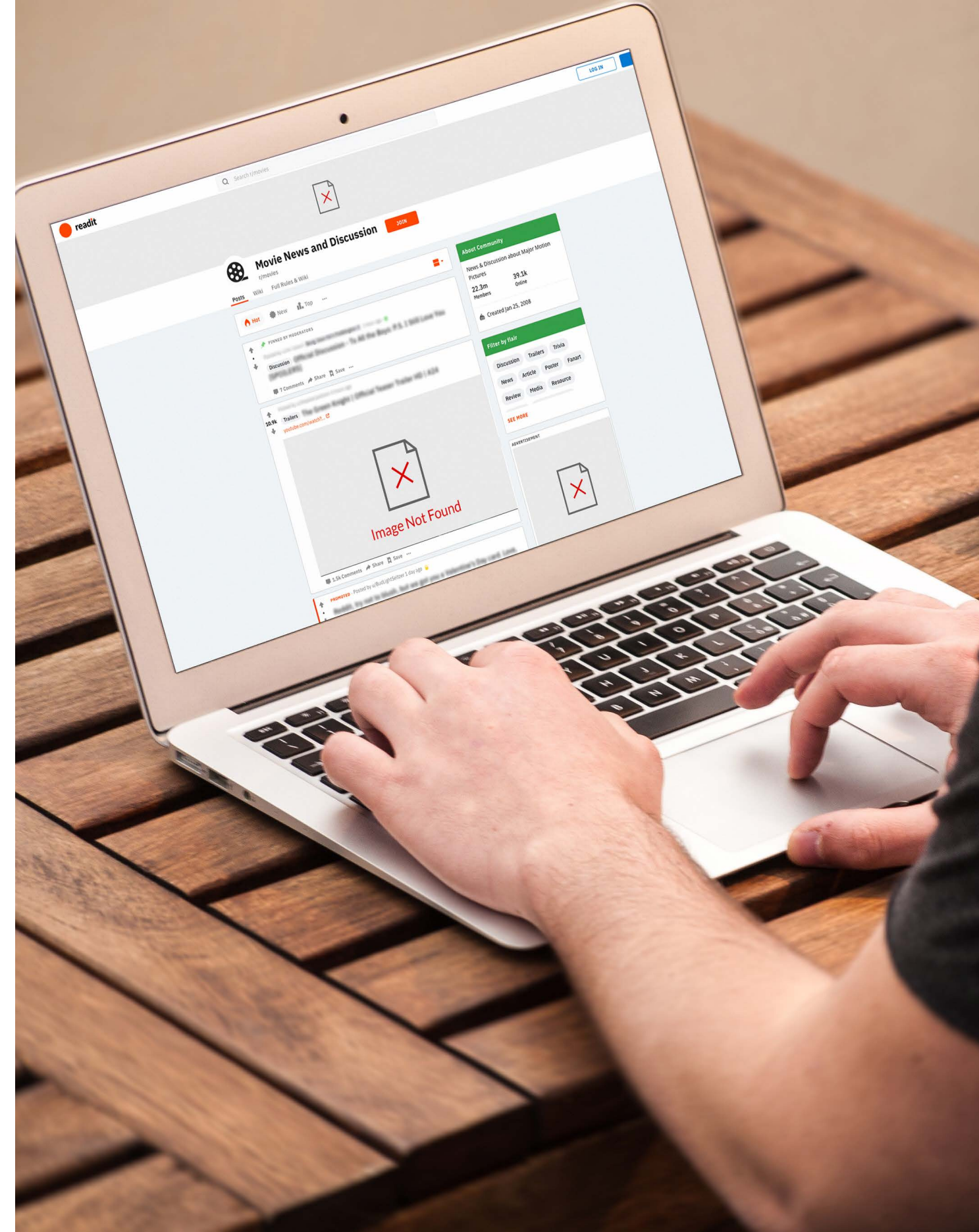
In this eBook, we dive into the technical stories behind several devastating API bugs that impacted five well-known large enterprises. Learn how “false positives” and other uncaught problems due to conventional testing and monitoring caused these companies to deploy unreliable and, at times, vulnerable APIs.



01

Every Object Is Important

Every character of an API's response can be critical to the overall system. An architect or engineer might think of API monitoring as akin to proofreading a textbook - a single typo could make important facts incorrect. The need to properly validate every object and data field is precisely what has led to the explosion in automated testing. However, functional and data errors can still slip past automated tests and monitors if they are not using enough test data. Coding can be difficult, and human error is common.



What Happens When a URL is “http://null”

A social media company was getting multiple reports of URL failures from their API consumers. URL problems can come in two flavors: error (e.g., encoding, double slashes, etc.) or failure (e.g., absolute vs. relative, URL vs URI). Unfortunately, they could not reproduce the reported issue, and as a result, couldn't decipher whether their URL problem was due to an API bug or the developers themselves.

When the social media company used API Fortress to create proper unified functional testing, they realized that instead of the URL being properly formatted or “null,” it would sometimes return an “http://null.” Developers using the social media company's API

platform expected to see only a “null” or a URL. By starting the field with HTTP, the “null” inadvertently tricked apps into interpreting it as a proper URL.

While it ended up being a sporadic issue related to content being uploaded into their database incorrectly, the company's API ecosystem lost much in tangible and intangible costs over the span of almost three weeks. Their API did not have a Swagger or OAI spec, and even if it did, a contract test would not have captured this issue.

SOLUTION:

Unify Functional Testing & Functional Monitoring

One of the unique advantages of API testing versus traditional UI testing is that it can process and validate the entire payload. This includes the data associated with each object. It is a form of data validation, and when combined with data-driven testing, that level of large scale testing can capture edge cases where some data has been put into the system incorrectly. In this case, the developer of the posting system had a bug that allowed an “http://null.”

Every character of an API's response can be critical to the overall system.

Incorrect Status Codes and Soft Error Codes

HTTP status codes that convey the results of a client's request are organized into five general categories. The 2xx response category indicates that a client's request was accepted successfully. Here are three examples:

200 OK: The HTTP request was successful.

201 Created: The accepted request resulted in a resource created inside a collection.

202 Accepted: Although the request was accepted, processing has not yet been completed.

Frequently, a developer will design APIs to return a 200 OK as a default, but a 200 actually means the operation completed successfully. What if the API call simply initiated a process that will complete later. That should return a 202, and API clients rely on this fact. Therefore, using anything besides a 202 can lead to bad assumptions.

Even worse, incorrectly returning a 200 instead of a 404 can be incredibly problematic. Ping tests will not check to see whether a 200 OK is actually a soft 404 because a web page is missing. False 200 OK issues can pose one of the most elusive and expensive threats in terms of costs and damage to reputation, due to how long the problems can persist before being caught.



When Open Banking Is Not 200 OK

After adopting open banking, a large bank was happy with the spike in popularity of their new financial services app for business and personal banking users. The bank's customers were glad to be able to easily transfer funds and make payments for various services on their mobile phones. Product leaders at the bank decided to enhance the personalization of the app by integrating an algorithmic feature that analyzes a user's spending to suggest how much they can save each month.

Months after launching the new recommendation service, the bank was disappointed to see the incredibly high drop rate. Customer service leaders at the bank became proactive and surveyed customers about the recommendation service only to learn that customers didn't like the service because it "lied" to them.

When customers activated the service, they were initially excited about the ability to simply push a button (YES or NOT NOW) to automatically transfer recommended savings amounts to their savings or investment accounts. After pushing the button, the API would respond with a 200, even if in reality the request was simply put in queue but should have returned a 202. Therefore, the app was misinforming the user by reporting a transaction was complete, when it was actually in-progress.

The reality is that no funds had been transferred, nor would anything eventually be transferred if there was a problem or error. **The application was programmed with status code standards in mind, but the API was not.** This led to consumers being misinformed and surprised to find out days or weeks later that a critical transaction was not executed.

SOLUTION:

Monitor for "Functional Uptime"

Modern API testing must provide insight about the entire API user flow. It is the best means to detect and diagnose one of the most prevalent API bugs - false positives. In this case, it would have been critical to not only post a transaction, but then also to confirm the transaction was executed. That entire flow should have comprised a monitoring test.

For more information about supporting SLAs with "functional uptime," read this [case study](#) about a large live sporting and entertainment events company. See how their monitors go beyond uptime and performance.

API testing must also provide insight about the entire API user flow

03

Memory Leaks

One of the most insidious reasons for problems with API availability and performance involves memory leaks - failures to release memory for allocation to execute processes. Memory leak errors are pervasive. Even a stateless (and cacheless) API executing standard CRUD on high traffic volume can instigate a memory leak error, requiring a server restart. Memory leak errors are potentially devastating, and can bring down business-critical processes that are either running or designed to run without termination.



A Slow and Steady Leak

A ticketing company had decided to launch a new partner API on the first of January. The new API would make it easier and more cost-effective for partner venues to sell tickets for their sports, arts, and entertainment events. It would also help their partners improve customer experience by extending branded experiences from pre-purchase to after-purchase.

With SLA adherence and uptime as the highest priorities for their partners, the ticketing company chose API Fortress to monitor the functional uptime, to act as a third-party audit of their uptime SLA guarantees. Just a week before the API was set to go live, they called us for a meeting. They had created a series of functional monitors, including one using a robust data set.

On the call they asked us, “Does your platform do anything weird when calling an API?” We were confused, and after some discussion they finally admitted the problem: their systems were running slower and slower the more scheduled tests that were executed. When we notified them that the platform makes normal HTTP API calls, they realized the issue - they had a memory leak.

While the API was stateless, the web server was not, and it continued to generate a new session with each transaction. This is a misconfiguration of the web server. The interesting part is that a load test would have discovered the leak rapidly, but since they were just scheduling a single monitor to run every 5 minutes, it took a while for the problem to reveal itself.

SOLUTION:

Conduct Performance (Load) Testing

In Forrester’s [Now Tech: Shift-Left Performance Testing \(SLPT\) Suites of Tools](#) report, analysts recommend “improving continuous testing with SLPT tools” to gain important insights for developers and testers about speed and load performance early in the lifecycle. However, most SLPT tools would have likely failed to diagnose the ticketing marketplace’s memory leak error during the design stage of API development due to an inability to unify integration testing with load testing.

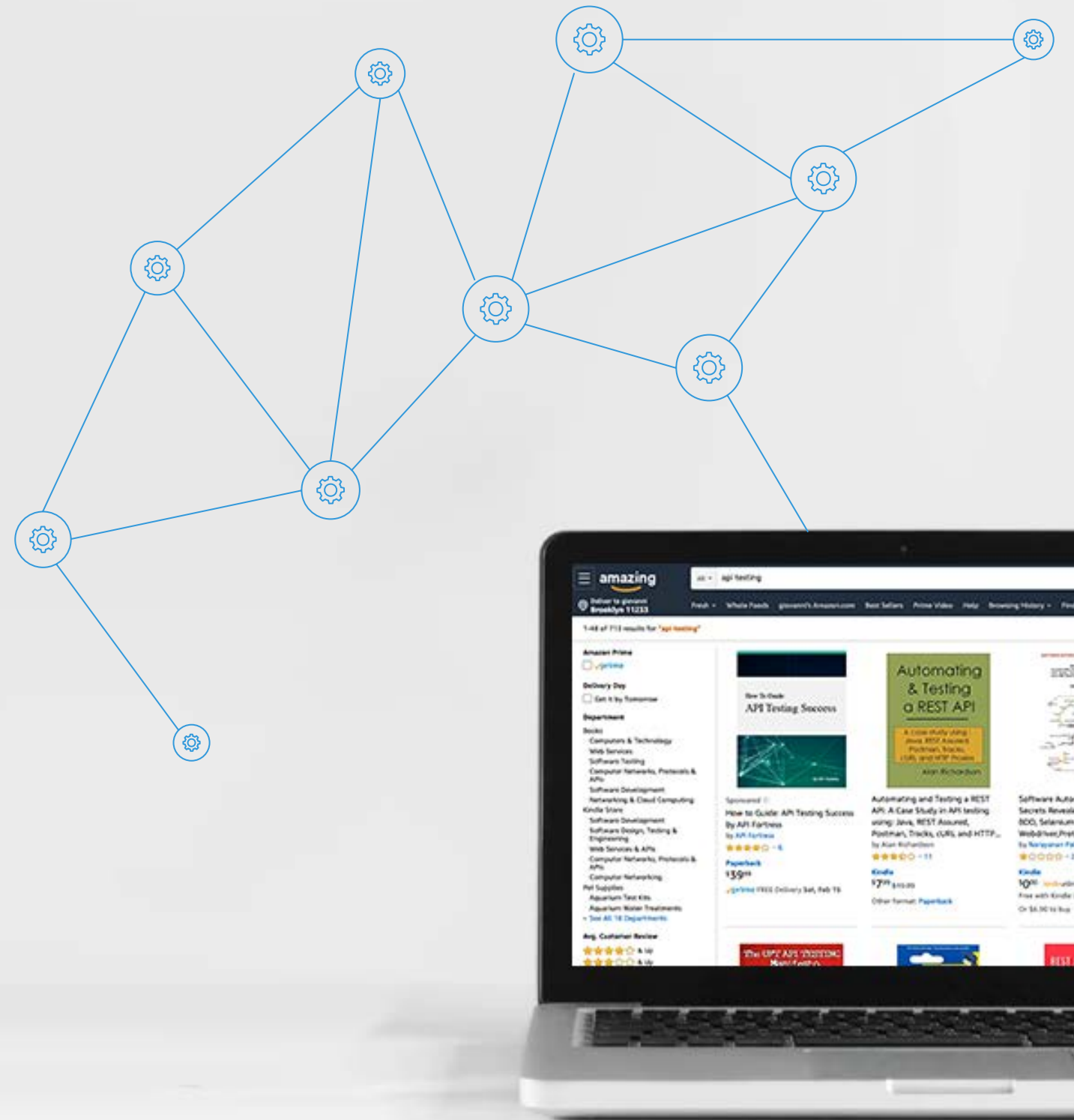
When the ticketing company deployed API Fortress, they were able to run their comprehensive functional tests as load tests. This helped them detect problems like memory leaks. The unified functional load test also helped them to accelerate diagnosis of the memory leak for rapid remediation. Ultimately, not only did the ticketing company solve their API problem, they significantly improved their posture in preventing a Denial of Service (DoS) attack.

Run a Load Test as early as possible

04

Outdated Caching

The cache is a vital component in the drive to maximize performance. This is true for personal computers, servers, and API managers. One of the best features of an API manager is to cache responses that are often hit. This is done for endpoints with information that doesn't change based on the user. That said, data does eventually change, and it is important to have policies in place to ensure that the pursuit of performance does not undermine functionality. Unfortunately, gateway teams are not always aligned with the operations and development teams in adherence to this goal, and that's exactly what happened in the next story.



404 Errors with No Endpoint or API Issues

A large global digital and print media publisher boasted of a nearly 100% uptime of their partner APIs over the last three months. In an effort to enhance the accountability of their service life agreements, they chose to trial API Fortress. They on boarded their two main endpoints - ProductListing and ProductDetails. The former showed all products in print, and the latter allowed you to dive into Product IDs individually.

Within four days, they were alerted to hundreds of 404 (“Not Found”) errors with the ProductDetails API. This mystified the publisher’s testing team, as they did manual testing to check and it always seemed to work.

In order to boost performance, the publisher’s API management team had cached the ProductListing API. It stored all Product IDs of items that are in print and for sale.

With API Fortress, they were able to create data-driven integration tests that first hit the ProductListing API, and then randomly selected hundreds of products IDs. In the subsequent step of using ProductDetails with the random product IDs, the platform discovered a problem that had plagued their partners for months.

Every Monday morning, they updated their Products database. An operation that involved the inclusion of new products, and the removal of others. This meant that their cached ProductListing API had hundreds of bad and missing IDs for hours until the cache refreshed. This was a time-sensitive API failure that could only be found by using one API as the source of the subsequent call.

Today, the publisher continues to monitor their API programs for “functional uptime” with API Fortress.

SOLUTION:

Use Integration Tests as Uptime Monitors


With detailed “functional uptime” reporting from API Fortress, the publisher was able to finally observe the caching error caused by the API management platform. The key is to not just exercise endpoints, but really consume the API like your users or partners would. A combination of using large data sets and integration testing allowed a monitor to capture a long-standing operational issue.

Proper API testing can capture everything from database issues, to API manager misconfigurations


Misaligned Documentation

API developers, documentation writers, and users (e.g. app developers) are often siloed into three different teams. This makes cross-functional communication of any “improvements” or other changes pivotal for continuous quality. Breakdowns in communication between the silos typically stem from not working in parallel throughout the lifecycle.

Even when closely following their QA processes, siloed teams can fall into traps in which their testing processes only verify their own team objectives. But problems caused by communication failures, or poor documentation, require a more collaborative approach. While DevOps processes and tools may help to bridge communication gaps, siloed teams should unify their testing tools with a proper DevOps toolchain. This requires workflows to become more transparent and collaborative.



API Fortress V3
 apifortress • apifortressv3


Apiary Powered Documentation
[Sign in](#) with Apiary account.

Documentation
Inspector

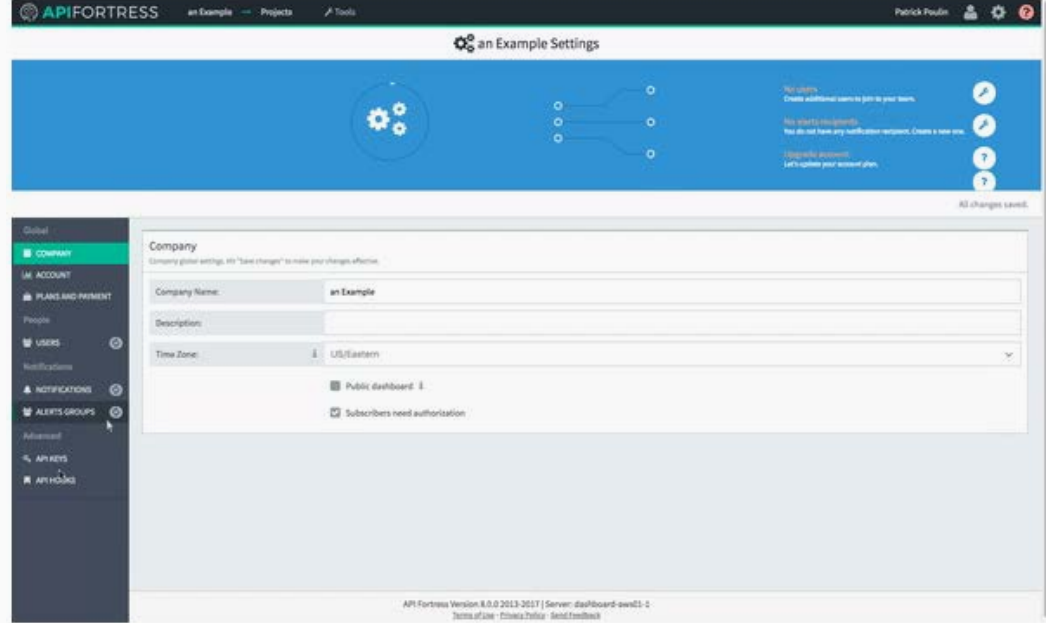
INTRODUCTION
REFERENCE

Authentication
Validators
Project
Tests
Test Run
Insights
Load testing

API Fortress V3

INTRODUCTION

Welcome to the new API Fortress API, which introduces the concept of **project hook**. To get started, you will first need to create a **hook** for the project you want to interact with.



- Go to your company settings page (with manager privileges)
- Click **API Hook**
- Click **+ Hook** to add a new hook.
- From there choose a project and user.
- Copy the hook to your **clipboard**

The generated URL will give you the ability to query resources of the connected project without needing any further authentication, so **mind who you share it with**. If you feel the url has been **improperly shared**, you can **revoke** it from the page you used to create it.

When UX and Development Are Not in Alignment

A large retail chain's e-commerce analytics team noticed a problem in sales - new earrings being listed were not selling, yet older listings of earrings had seen virtually no dip in sales. The retailer's tests and monitors showed that nothing seemed to be wrong with their APIs and platform. However, some manual testing revealed that the new listings were missing from the store's top-level navigation.

With the help of API Fortress, hundreds of detailed functional tests were run against their APIs. The QA team quickly found that new earrings were being categorized as "earrings," but they had

been categorized as "jewelry" up to that point. A supply manager had created a new category ("earrings"), but didn't realize the impact it would have on the UX team: the front-end team was siloed, using only what they knew about the category options for the top-level navigation.

While lost revenues from coding errors are bad enough, this example of lost revenues was especially frustrating as it came from a new employee actually being proactive about making a change they viewed as beneficial.

SOLUTION:

Standardize API Testing & Monitoring Across All Teams

By working in silos, the different teams at the retailer had pursued different goals. While the UX team desired to make it easy to find "jewelry," the team in charge of uploading items to the platform created a category for which the API team and UX team had not been prepared.

As an API-first platform that integrated easily into the retailer's DevOps toolchain, API Fortress allowed the siloed teams to standardize all testing and monitoring via a centralized platform. Stakeholders of all coding abilities could understand and be notified when something unexpected was happening.

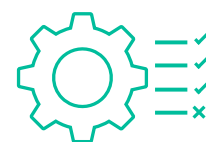
Sometimes, trying to do good can lead to bad results.

Conclusion: Checklist for Functional Uptime Monitoring

“We noticed a considerable increase in the quality of code released with the help of API Fortress. Their flexible platform makes it easy to integrate API testing into any workflow without disrupting legacy or new tools. We shifted testing left and standardized a single API testing and monitoring strategy across the whole organization.”

—Sander Rensen, Lead Solution Architect at CapGemini

As you implement API testing and monitoring, make sure you consider the API test habits that would capture these bugs before they have an impact on your bottom line.



Create

Build detailed API functional tests, recreate entire user flows



Monitor

Reuse those tests as monitors of production and important internal environments



Standardize

Give access to notifications and reports across teams



Automate

Data-driven testing is the best way to find even the most elusive, but costly, bugs

Start your [Free Trial](#) of API Fortress with Unlimited API Monitoring. Or [Schedule a Demo](#).