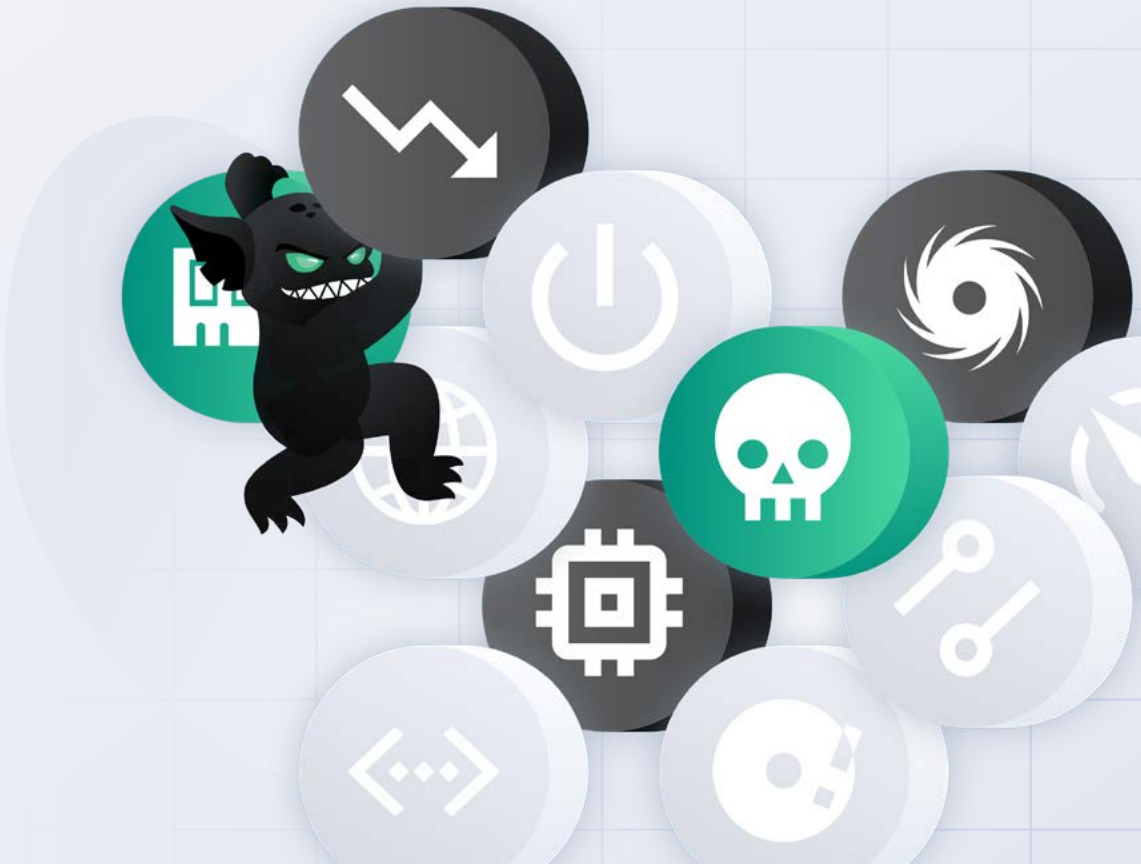
EBOOK

Getting started with Gremlin attacks

HOW TO USE GREMLIN'S ATTACKS IN A MATURE CHAOS ENGINEERING AND RELIABILITY PRACTICE



Gremlin

Table of Contents

Introduction

Gremlin provides a range of unique attack types to help you test your systems. Each attack impacts specific infrastructure resources, such as CPU consumption, storage device throughput, running processes, and network bandwidth. By running these attacks on our systems, we can uncover unexpected behaviors, validate resilience mechanisms, and improve overall reliability.

The goal of running an attack isn't to "create chaos," but to use them as a means to identify and fix failure modes. This is why we recommend defining your reliability goals, planning out your experiment, and ensuring you have visibility into your systems before you press the "Unleash Gremlin" button. Because Gremlin offers such a wide range of attack types, each with their own configurable options and behaviors, it helps to understand how each attack works and when you would want to use it.

In this white paper, we'll explain each of Gremlin's attacks in complete detail. We'll explain what each attack does, how it affects systems, and provide best practices for configuring and running it. We'll also present common technical and business initiatives that each attack helps solve, so that you can directly apply your reliability work to your organization's goals. By reading this white paper, you'll learn:

- **How to run each of Gremlin's attacks.**
- **What each attack does, the failure modes that it can help identify, and the resilience mechanisms it can help validate.**
- **When you would want to use one attack over another.**
- **How to interpret the results of an attack and use this knowledge to improve reliability.**

Intended audience

This white paper is intended for practitioners who will be directly involved in creating and running experiments within Gremlin. This includes site reliability engineers (SREs) and other technical engineers responsible for the reliability of an application, service, or infrastructure component. This also includes any stakeholders involved in running a GameDay.

What is an attack in the context of Chaos Engineering?

An attack is a method of creating or injecting stress and/or failure into a part of a system, such as causing network disruptions or exhausting resources. This is also known as fault injection. Attacks are the means by which we perform chaos experiments and Chaos Engineering in general, as they allow us to identify failure modes in systems.

Attacks that directly target infrastructure resources are called infrastructure layer attacks. These attacks impact systems directly by targeting system-level resources.

These include:

- **Resource attacks, which generate load across CPU, memory, and storage devices.**
- **State attacks, which change the state of the environment that your systems are running in.**
- **Network attacks, which simulate unhealthy network conditions including dropped connections and outages.**

Infrastructure attacks are executed by an agent running on a host. The agent initiates attacks, manages them throughout their lifespan, and safely stops them once they're complete. The agent can also halt a running attack and roll back its effects at any time. The impact of the attack will only be visible as long as the attack is running. Once the attack completes (or if it's halted), the impact is reverted and the system returns to its normal operating state.

At this point, you may be thinking, "Why should I inject stress or failure into my system? We have enough 'chaos' already." While injecting controlled stress or failure into a system may seem counterintuitive, it's the best way of ensuring that a system is resilient. When teams build complex systems, it's impossible to anticipate the many different ways these systems can fail in high-stress real-world scenarios. This is where Chaos Engineering and attacks fit in. Teams use attacks to simulate a wide range of stressors acting upon their systems. This helps catch and uncover vulnerabilities before they become production outages, which can result in serious technical and business consequences as shown on the next page.

COST OF DOWNTIME IN REVENUE/MIN

Target	Walmart	Amazon
$8,476/minute	*$40,771/minute*	*$220,000/minute*

COST OF RESOLUTION

$18.2M/Year

Is the cost of just the resolution of reliability-related outages. Assuming 280 hours of downtime a year (97% uptime), 5 hours to find and fix the problem, and 50 employees involved in resolution at average wages.

IMPACTS ON CUSTOMER EXPERIENCE

37%

of downtime-related lost revenue is due to reputation damage and customer churn as a result of poor customer experience.

44%

of customers are likely to tell others about a bad experience.

VALUE OF IMPROVING UPTIME

If downtime costs $10,000/min, improving uptime results in the following gains in revenue per year:

99% → 99.9%	*$47,154,000*
99.9% → 99.99%	*$4,735,800*
99.99% → 99.999%	*$473,600*
99% → 99.999%	*$52,363,400*

Understanding each attack type

Gremlin offers 11 different infrastructure layer attacks. These attacks are separated into three categories: resource attacks, state attacks, and network attacks. The following table shows each category and its corresponding attacks:

Resource attacks	State attacks	Network attacks
CPU	PROCESS KILLER	BLACKHOLE
DISK	SHUTDOWN	DNS
I/O	TIME TRAVEL	LATENCY
MEMORY		PACKET LOSS

Throughout the rest of this white paper, we'll explain what each attack does and present specific use cases. We'll map these use cases to technical initiatives, then tie these to higher-level business objectives. This way, you will have the information and guidance you need to develop a Chaos Engineering strategy that best meets your team and organizational goals.

Resource attacks

Resource attacks generate load across computing resources including CPU, memory, and storage. This lets you prepare for sudden changes in load, validate autoscaling rules, test monitoring and alerting configurations, and make sure your systems are stable even under heavy demand.

CPU attacks

DOCS | BLOG | WEBINAR

CPU attacks are often called the "Hello World" of Chaos Engineering, referring to introductory programming exercises that simply print the phrase "Hello World". While CPU attacks are simple and a great starting point, unlike a "Hello World" program, they're also extremely useful for real-world applications.

Although modern computers typically have multiple cores per CPU (and for higher-end systems and servers, multiple CPUs) in order to run multiple concurrent operations, they still have limits. These limits are easy to hit in normal, everyday operations—for example, when multiple applications compete for resources (often called the "noisy neighbor" problem). CPU attacks help you ensure that your application behaves as expected even when CPU capacity is limited or exhausted. CPU attacks can also help test and validate automatic remediation processes, such as auto-scaling and load balancing.

CPU attacks can answer questions such as:

- **How is the user experience impacted when CPU resources are exhausted?**
- **Do I have monitoring and alerting in place to detect CPU spikes?**
- **Do I have quotas configured to limit CPU by application/process/container?**
- **Do I have cleanup scripts to get rid of corrupted threads?**

Technical use cases

Ensure stability and performance under stress

Heavy traffic can place a high load on systems, but our systems must remain operational even when the CPU is stressed. Performance testing tools are often used to recreate heavy load, but we can use CPU attacks to create the same impact. This is also useful for preparing for noisy neighbors when migrating from a dedicated data center to a shared environment, such as a public cloud.

Validate autoscaling and alerting mechanisms

Cloud platforms and orchestration tools can automatically scale up applications and systems to meet increased demand. In addition, monitoring and alerting tools can notify you when capacity gets low. However, it's up to engineers to set these thresholds and configure these mechanisms. CPU attacks let you trigger scaling thresholds to make sure your applications can scale quickly and effectively, as well as trigger alerts to make sure your alerting mechanisms are configured properly. Triggering autoscaling mechanisms also lets you ensure that you can reliably migrate workloads to other nodes and load balance traffic.

Validate resource quotas

With containers, resource quotas limit the resources that are made available to any one container. This is useful for preventing rampant consumption caused by runaway processes, while ensuring that each container has enough resources to run effectively. CPU attacks let you test these quotas and understand how your applications respond when they exhaust their available resources. It also lets you ensure that your orchestration tool enforces quotas as expected. CPU attacks let you test these quotas and understand how your applications respond when they exhaust their available resources. It also lets you ensure that your orchestration tool enforces quotas as expected.

Business objectives

Prepare for peak traffic events

Increasing CPU usage simulates the effects of high demand that would normally only be experienced during a high traffic event like Black Friday or Cyber Monday. CPU attacks let you safely simulate the impact of these events in a controlled environment to better prepare for them.

Streamline cloud migrations

Before starting a cloud migration, use CPU attacks to simulate cloud conditions such as noisy neighbors on shared infrastructure, or instances with reduced CPU capacity. This will help prevent unexpected issues from arising once you start the migration process.

Reduce operating expenses

It's easy to over-provision CPU capacity based on expectations of production traffic, especially in a cloud environment. CPU attacks can help you measure the performance of your applications under various levels of CPU demand, giving you the necessary information to adjust capacity. Validating autoscaling also makes it possible to dynamically adjust capacity in real-time, further reducing costs.

Memory attacks

DOCS | BLOG | WEBINAR

Memory (or RAM, short for random access memory) is a critical resource that determines how many processes can run simultaneously on a system, as well as how much data those processes can work with at any given time. RAM is essential to both system stability and performance. While it's possible to exhaust CPU capacity without significant consequences, exhausting RAM can cause a system to lock up, processes to close, and create significant disk activity due to memory paging (also known as thrashing).

The Memory attack consumes a set amount of RAM, either as a specific amount (in MB or GB) or as a percentage of total system memory. By purposefully consuming memory, you can test your systems to ensure they can tolerate a sudden increase in usage, and to gauge the impact on performance.

Running Memory attacks can answer questions such as:

- **How does application and system performance/stability degrade as memory resources become scarce?**
- **Do we have alerting set up to detect when memory becomes scarce? Is it working?**
- **How does the system handle low-memory or out-of-memory (OOM) situations?**
- **Does it crash? Does the "OOM Killer" process start closing applications?**
- **When the application degrades due to memory loss, is any user data lost?**

Technical use cases

Prepare for memory leaks and out of memory (OOM) errors

Memory leaks happen when applications gradually consume more memory than they release. Over time, this can consume enough memory to cause system instability, crashes, and swapping (also known as paging) which can significantly impact application performance and availability. This can also trigger out of memory (OOM) errors, where the system has no available memory left to allocate and must start terminating processes.

Test memory-intensive workloads

Some workloads are inherently memory-hungry, such as in-memory caches, databases, and machine learning models. Using Memory attacks lets you simulate the impact these workloads would have on your systems before you deploy them, helping you with capacity planning and right-sizing your infrastructure.

Prepare for a cloud migration

Like CPU attacks, memory attacks can help you prepare for cloud migrations by simulating noisy neighbors, setting and validating autoscaling thresholds, validating the configuration of resource quotas (such as cgroups and Kubernetes resource limits), and validating alerting thresholds. This way, you can test your workloads under cloud-like conditions before migrating them.

Business objectives

Prepare for peak traffic events

Like CPU, memory usage can change in response to increased demand. Performing memory attacks lets teams simulate these conditions in advance of high traffic events like Black Friday or Cyber Monday so that you can better prepare.

Reduce operating expenses

Right-sizing memory capacity can save significantly on infrastructure costs, especially in the cloud. Memory attacks allow teams to validate the stability of their applications under changing memory conditions, allowing for better capacity planning and potential cost savings.

Streamline cloud migrations

Before starting a cloud migration, use memory attacks to simulate cloud conditions such as noisy neighbors on shared infrastructure, or instances with lower memory capacity. This will help prevent memory exhaustion during or after a migration.

Disk attacks

DOCS | BLOG | WEBINAR

Disk space is a necessary resource for persistent data such as operating system files, database records, configuration files, and logs for troubleshooting and debugging. Disks also perform critical operational functions, such as providing swap space for virtual memory, which prevents system crashes due to out-of-memory errors. Modern systems typically use solid-state drives (SSDs) instead of more traditional hard disk drives (HDDs), but the word “disk” refers to all kinds of persistent storage.

Disk attacks will help you determine how your systems and applications behave when storage space is limited or unavailable. They can also help validate dynamic storage provisioning systems, such as database sharding and persistent volumes in Kubernetes.

Disk attacks can answer questions such as:

- **Do we have alerts set up to tell us when disk usage is close to capacity? Is the lead time long enough?**
- **When the disk is almost full, do we know how to mitigate the situation?**
- **Do we have mechanisms or policies in place to add capacity when we run low?**
- **Do our applications fail when the disk is full? Is user data lost as a result of this failure?**

Technical use cases

Test disk capacity and throughput

Disk attacks are often used to simulate reading or writing a large data set, such as a restored backup, replicated database, or large log file. Not only does this let teams verify disk capacity before migrating a workload, but the process of filling in disk space also impacts performance. If an application reads or writes large amounts of data, this can cause disk throughput to drop, which in turn causes application throughput to drop. A large disk attack can therefore test capacity, throughput, and concurrency simultaneously.

Test automatic disk cleanup, compression, and provisioning

Teams use several automated mechanisms to add disk capacity. Automatic cleanup and compression tools can prevent disks from filling too quickly, such as log rotation. Some systems also provide dynamic storage provisioning (e.g. sharding) to dynamically increase disk capacity as demand increases. Disk quota systems limit the amount of disk space used by specific applications or processes. Disk attacks let teams test these mechanisms and ensure they work as expected.

Validate monitoring and alerting mechanisms

As with other resources, teams should monitor disk usage to avoid problems caused by running out of disk space. Once you've set monitoring and alerting thresholds, use a disk attack to ensure that teams are notified of low capacity early enough to respond to and mitigate the issue.

Business objectives

Reduce operating expenses

Persistent storage can be particularly expensive in cloud environments, especially for faster, higher-capacity types of storage. Disk attacks let teams simulate the effect

of less expensive storage devices on their workloads, which can save money without compromising performance.

Reduce the risk of outages

Disk exhaustion can lead to unexpected outages. By using disk attacks to proactively test monitoring and alerting capabilities, teams can potentially eliminate disk usage as a cause of high-severity incidents and outages.

IO attacks

DOCS | BLOG | WEBINAR

Reading and writing to disks often remains the slowest part of most applications, especially when using spinning disks. As applications handle larger volumes of data and perform more operations, the increased IO (short for input/output) can have a magnifying effect on latency and even lead to timeouts and incidents. It can also impact your CPU resources.

IO attacks allow you to simulate heavy IO operations and view their effect on your applications. With this knowledge, you can build more reliable applications by optimizing your systems for faster IO (e.g. moving to SSDs), using caching mechanisms to avoid disk interactions, or performing heavy read/write operations in memory while periodically persisting to disk.

IO attacks answer questions such as:

- **What happens when the application can't write to disk due to latency? Is the user forced to wait? Do they need to resubmit their request?**
- **How much disk latency can the system handle before data becomes corrupt or unavailable?**
- **Do we have alerts in place to detect IO bottlenecks? Do we know how IO impacts other performance issues, such as high request latency?**

Technical use cases

Prepare for a storage migration

Changing storage devices can have a significant impact on latency. SSD and NVMe devices are much faster than traditional drives (HDDs), but cost more, especially in a cloud environment. Migrating to the cloud also often means migrating data to higher-latency storage solutions, such as network attached storage or storage services. Each storage location has a different number of IOPS (input/output operations per second), which affects application performance. An IO attack can help you prepare for slower storage solutions by simulating their performance. This lets you measure the impact of

slow storage on the user experience and determine the cost benefit of faster storage devices before starting a migration.

Test disk-heavy workloads

Some applications require frequent or heavy disk access, such as databases. Heavy IO operations can also impact CPU and RAM usage, creating unintended side effects for applications and systems. IO attacks let you simulate these applications and operations, and observe their impact on your systems.

Validate the effectiveness of a cache

Caches such as Redis can reduce IO activity by storing some data in memory. Some systems also use delayed write caching to temporarily store data in memory before writing to disk in order to increase performance. With an IO attack, you can validate the effectiveness of a cache by stressing the underlying disk and measuring the responsiveness of your application. If the application and cache are configured properly, there should be little to no impact on performance.

Caching can also protect you from storage that fails or becomes unavailable. For example, a cloud storage device failure led to data being unavailable and in some cases lost. Using an IO attack won't cause a disk failure, but it can simulate an unavailable disk by consuming all available IOPS and making the disk unresponsive. This lets you prepare for unexpected disk outages and ensure your cache is working effectively.

Business objectives

Prepare for peak traffic events

During a peak traffic event like Black Friday or Cyber Monday, disk-heavy services like product catalogs and order fulfillment will need to access more data stored on disk, reducing throughput. Running an IO attack can simulate this effect so teams can better prepare for high demand in production.

Improve the end user experience

Disk latency can have an impact on the user experience (UX). Testing application performance during an IO attack can pinpoint areas of latency and allow application developers to better optimize the application.

Streamline new product launches

High-bandwidth services like media streaming and file sharing applications often require high disk throughput. When preparing to launch these and similar services, use IO attacks to ensure that your services are stable and performant in periods of high demand.

State attacks

State attacks change the state of your environment by terminating processes, shutting down or restarting hosts, and changing the system clock. This lets you prepare your systems for unexpected changes in your environment such as power outages, node failures, clock drift, or application crashes.

Process killer attacks

DOCS | BLOG | WEBINAR

Process killer attacks allow teams to terminate a specific process or set of processes. This is mainly done to prepare for application crashes, out of memory (OOM) events, and similar events.

Teams use this attack type to answer questions such as:

- **If a key process dies, such as the httpd process, what happens to our website? Does it become entirely unavailable?**
- **Do we have recovery mechanisms, and if so, do they detect and restart the killed process?**
- **When a process running an application (e.g. Tomcat) terminates, what percentage of user traffic is affected, and for how long?**

Technical use cases

Testing automatic restarts

Watchdog processes like systemd can automatically detect and restart failed processes. Container orchestrators like Kubernetes perform a similar function by detecting and restarting failed containers. Process killer attacks can confirm that:

- **The watchdog is properly configured to detect and restart the killed process.**
- **The detection threshold is low enough to prevent lengthy outages.**
- **The process successfully restarts with no data loss or corruption.**

Test leader re-election for clustered workloads

Clustered workloads like Kafka or Kubernetes require one or more nodes to act as leaders in order to maintain a consistent cluster state. If a leader fails, another node is automatically elected as the new leader. We can ensure this process works as expected by running a process killer attack on the Kafka or Kubernetes process on a node, then validating that a new leader is elected and that cluster state is maintained.

Business objectives

Maximize the availability of a service

Process killer attacks help teams maximize the availability of a service. Teams can use process killer to test automatic recovery processes and ensure that services continue operating without disrupting the user experience. Not only does this reduce downtime, but it reduces the risk of a poor user experience.

Shutdown attacks

DOCS | BLOG | WEBINAR

Shutdown attacks shut down (and optionally reboot) the host operating system. This is much like how Chaos Monkey, an early Chaos Engineering tool created by Netflix, works. Shutdown attacks let teams build resilience to host failures by testing how their applications and systems behave when an instance is no longer running.

This answers questions such as:

- **How long does it take for an instance to restart? Does my application successfully restart when the instance comes back online?**
- **Does my load balancer automatically reroute requests away from the failed instance? Do I have other instances available to handle these requests?**
- **If a user has an active session on an instance that fails, does the session gracefully continue on a different instance?**
- **Is there any data loss? Are ongoing processing jobs restarted?**

Technical use cases

Ensuring workloads migrate successfully

When a clustered node starts its shutdown process, applications and services running on it should automatically migrate to a healthy instance. This maximizes uptime and availability, while also reducing the risk of data loss or corruption. You can use shutdown attacks to ensure that:

- **Applications and services halt on the target instance and are automatically replicated to healthy instances.**
- **Any active connections or user sessions are closed without interrupting service.**
- **Load balancers automatically route traffic away from the terminated instance.**

Validating high availability of clustered workloads

Clustered systems such as Kubernetes and Kafka are designed to handle node failures, but it's important to validate that these mechanisms behave as expected. There are situations where losing a node can cause complex problems. For example, losing more than half of the nodes in a cluster can cause cluster state to diverge between the remaining nodes, resulting in an inconsistent view or "split brain." We can use a shutdown attack to halt one or more nodes and ensure that the cluster continues to operate without entering a split brain.

Validating automatic node restarting or replication

Nodes can automatically restart for a number of reasons including operating system updates, scheduled reboots, or requests from the cloud platform. Using shutdown attacks lets you recreate these scenarios to ensure that you can tolerate an unexpected shutdown or reboot of a node. This lets you:

- **Test mechanisms in your cloud platform to detect and address shutdown nodes.**
- **Measure how long it will take to restart a failed instance as part of capacity planning.**
- **Determine the cost-effectiveness of using short-lived instances.**

Business objectives

Maximize the availability of a service

Like process killer attacks, shutdown attacks help teams maximize uptime by ensuring that automatic replication and failover processes are functioning correctly. This lets teams mitigate interruptions in service and provide a strong customer experience. In addition, shutdown attacks can help you right-size your infrastructure by simulating the use of less expensive, short-lived cloud instances.

Time travel attacks

DOCS | BLOG | WEBINAR

Time travel attacks allow you to change the system time. This lets you prepare for scenarios such as Daylight Savings Time (DST), clock drift between hosts, and expiring SSL/TLS certificates. Teams run these attacks to determine:

- **How does our application behave if the clock skews by seconds? By minutes?**
- **Do we have an automated process that can handle expiring SSL/TLS certificates?**
- **Is any data lost or corrupted when our master and read replica databases don't have the same timestamp?**

Technical use cases

Prepare for Daylight Savings Time (DST) and leap years

Clock changes caused by Daylight Savings Time and leap years can cause communication problems between systems and applications, resulting in discarded messages or incorrect timestamps. These events often aren't thoroughly tested due to their infrequency, but can nonetheless have significant effects on time-sensitive systems. You can use a Time Travel attack to move the system clock forward or backward one hour, test your application, and verify the integrity of your data.

Ensure security by testing TLS certificate expiration

TLS certificates are a cornerstone of modern Internet security and depend on accurate timekeeping. Certificates expire after a certain amount of time and must be renewed and replaced, or else communication will be disrupted. Time Travel attacks let you test the impact of clock skew on encryption and make sure that you stay ahead of expiring certificates by testing notifications and automatic renewal processes.

Prepare for "end of epoch" problems

The systems that computers use to track time can have their own unexpected behaviors. For example, the Year 2000 problem (Y2K) was caused by computers storing the current year as a two-digit number, which meant that they couldn't differentiate between January 1, 2000 and January 1, 1900 as they were both stored as "00." Similarly, the 2038 problem is caused by a limitation of how 32-bit Unix-based systems store date and time values. On January 19, 2038, this value will overflow and roll the date back to December 13, 1901. If you're not sure whether your systems are protected, you can use the Time Travel attack to proactively test them by moving the date forward. You can use this same strategy to prepare for similar "end of epoch" events.

Test clock sync (NTP) between nodes

Systems that share data often require synchronized system clocks, which are best managed using a service like NTP. Mismatched timestamps can lead to problems like discarded messages and data conflicts. It's possible that NTP may be unavailable and your clocks will drift. Using Time Travel lets you simulate clock drift and NTP outages simultaneously so that you can proactively prepare for an outage.

Business objectives

Meet security and compliance requirements

Time tracking is essential to security and compliance. The encryption methods that allow for secure communications between your systems and your customers rely on clock synchronization, which is why mismatched system clocks can have a significant impact on system stability and usability. Time travel attacks allow teams to prepare for potential situations where clocks can become desynchronized, determine the impact on security and compliance, and mitigate this impact in advance of a production incident.

Network attacks

Network attacks let you simulate unhealthy network conditions including dropped connections, high latency, packet loss, and DNS outages. This lets you build applications that are resilient to unreliable network conditions.

Blackhole attacks

DOCS | BLOG | WEBINAR

Applications often have many dependencies (both known and unknown) to downstream systems and services. Although our systems are becoming increasingly dependent on networks, we can't guarantee network availability at all times. This forces us to think of ways to proactively compensate for situations where dependencies are unavailable or unreachable. Blackhole attacks let you simulate these outages by dropping network traffic between services. This lets you uncover hard dependencies, test fallback and failover mechanisms, and prepare your applications for unreliable networks.

Blackhole attacks can answer questions such as:

- **Where do dependencies exist within our system?**
- **Do we have monitoring in place to alert on the unavailability of each service?**
- **Does our application gracefully degrade if a dependency is unavailable?**
- **Is the user experience negatively affected when a downstream dependency is unavailable?**
- **Do we have dependencies that we think are non-critical, but can actually bring down our entire application?**

Technical use cases

Test dependency failures

Testing resilience to failed dependencies is crucial for modern applications. Since dependencies are outside of the control of engineering teams, applications must be designed to tolerate and work around dependency failures. Teams can use Blackhole attacks to validate that if a dependency fails, the application degrades successfully instead of crashing. This also lets you test error-handling mechanisms such as failover, error messages, and retries.

Blackhole attacks can also help determine whether a dependency is critical or non-critical. Critical dependencies are essential for an application to operate, such as databases, network storage services, and payment processing services. It's not always clear whether or not a dependency is critical, but by using a Blackhole attack, teams can quickly test each dependency to determine its criticality.

Validate cluster resiliency

Although clustered workloads are generally resilient to network outages, major disruptions can put cluster stability at risk. For example, if two nodes in a three-node cluster become disconnected, it could result in a split brain scenario. When the nodes come back online, they'll each have a different and inconsistent view of the cluster.

High Availability (HA) clusters can reduce the impact of outages by monitoring node state, providing redundancy, and handling failover. Blackhole attacks can validate that your HA cluster is able to gracefully handle unresponsive nodes, recover from failures, load balance traffic to healthier nodes, and avoid split brains.

Validate monitoring and alerting

Being able to detect unresponsive, disconnected, or failed nodes is critical to reliability. Your monitoring tools should be configured to detect nodes that are experiencing network outages and start an automatic remediation process or notify an engineer. With a Blackhole attack, you can test that your monitoring and alerting tools are configured to correctly notify your teams quickly.

Business objectives

Business continuity planning and disaster recovery planning

Blackhole attacks are commonly used to simulate a sudden loss of a system or service. If these systems are critical to your business, it's important to have a plan to restore service as quickly as possible. Blackhole attacks offer a safe and controlled way of

simulating system-wide outages, allowing teams to practice their incident response and recovery plans without the added stress of a real-world disaster.

Streamline a cloud migration

In the cloud, your availability is only as high as the availability of your cloud provider. If a network outage brings part of your cloud provider offline, it can impact your own systems. Using blackhole attacks lets you prepare for these scenarios in advance so that you can make your services more resilient and avoid unexpected downtime.

DNS attacks

DOCS | BLOG | WEBINAR

The Domain Name System (DNS) is a system for naming computers, services, and other resources connected to a network, such as the Internet or a private network. DNS associates domain names with IP addresses, allowing resources to reference each other using names instead of addresses.

The DNS attack simulates a DNS outage by blocking network access to DNS servers. This lets you prepare for DNS outages, test your fallback DNS servers, and validate DNS resolver configurations.

This answers question such as:

- **Do you have a secondary DNS server and do your services fallback automatically to it?**
- **Do you gracefully reroute calls back to the primary once it's back online?**
- **How do your services behave during a major DNS outage, such as the DynDNS or Akamai outages?**

Technical use cases

Prepare for DNS provider outages

Despite being a critical part of modern networks, DNS failures often catch teams unprepared. In July 2021, an Akamai outage caused many top websites to go down across airlines, retail, finance, and other industries. Running DNS attacks lets you proactively simulate a DNS outage to test your ability to fallback to other DNS services, reroute traffic to the primary service once it's back online, and ensure that your services remain operational during the outage.

Business objectives

Maintain service availability

Reliable DNS ensures that your customers can access your websites over the network. A DNS outage can make it appear as if your website or web services are offline, even if all of your systems are operating normally. Teams can use DNS attacks to prepare for and mitigate the impact of DNS outages, improving availability and uptime.

Latency attacks

DOCS | BLOG | WEBINAR

Latency is the amount of time taken for a network request to travel from one network endpoint to another. A number of factors can affect latency including the physical distance between endpoints, the number of routers and switches, the maximum throughput of the connection, and the speed at which both devices can process network data. With user-facing services like websites, lower latency contributes to faster load times, which has a positive impact on user experience and conversion rates.

The Latency attack injects a delay into outbound network traffic, letting you validate your system's responsiveness under slow network conditions. This helps maximize the performance and responsiveness of user-facing applications, improve the throughput of backend systems, ensure messages aren't lost, and test automatic load distribution across nodes.

Latency attacks answer questions such as:

- **What impact does network traffic have on latency? Do we have enough capacity to handle traffic bursts?**
- **How does latency impact the user experience?**
- **Can we reliably redistribute traffic from high latency nodes to low latency nodes?**
- **If there's a high latency event, what's our recovery time to return to a normal, real-time user experience?**

Technical use cases

Migrate to the cloud

Cloud environments can introduce more latency compared to on-premises data centers, especially when resources are spread out across multiple availability zones

or regions. Running Latency attacks before a cloud migration can help you determine how your application will perform in a higher latency environment.

Simulate unstable network conditions and poor connectivity

Modern applications are often designed with the assumption that users and cloud providers will always have stable, fast Internet connections. This isn't always the case: cloud provider networks can become saturated, users have data caps and bandwidth limits, and users may be located far from your hosting provider.

Latency attacks can recreate these conditions and show how your systems would realistically perform in a real-world scenario. You can then optimize your application to ensure a consistently good user experience across a wide range of network conditions.

Configure load balancing rules

Many network gateway and load balancing tools can automatically reroute traffic between nodes based on latency. For example, Amazon Route 53 can route individual DNS queries to the AWS Region that gives a user the lowest latency. You can ensure this is configured correctly by running a Latency attack on a specific node, availability zone, or even an entire region, and validate that traffic is rerouted to a faster resource.

Fine-tune timeout and retry thresholds

Without a well-tuned timeout, users could end up waiting an unknown length of time for a response from the server. The longer your service takes to respond to a user, the more likely they are to abandon you for a competitor. Timeouts ensure that requests take no more than the specified amount of time to process, and Latency attacks help you verify that this is the case.

Retry mechanisms let you repeat requests that fail or take too long to complete. This lowers the risk of a request failing due to short-lived errors, such as a temporary network disconnection. Like with timeouts, Latency attacks let you validate that your retry mechanisms are configured and working properly.

Test concurrency control

For time-sensitive workloads such as a messaging queue or distributed database, latency can cause problems with data integrity, distributed lock management, and replication. Latency attacks let you preemptively test these mechanisms to make sure they're resilient and won't result in unexpected behaviors.

Business objectives

Prepare for peak traffic events and improve the user experience

Peak traffic events result in a significant increase in network traffic, which can lead to increased network latency. In turn, this can cause instability and increased errors. Running latency attacks lets you proactively recreate high latency events, observe and mitigate the effects of latency on your systems, and be better prepared for when they happen.

Streamline a cloud migration

When migrating from on-premises to a cloud platform or service, network latency becomes a performance factor. Cloud infrastructure and services communicate entirely over the network, and if your services are distributed, the amount of latency increases. Running latency attacks on-premises allows you to observe the impact of additional latency on your applications before beginning a migration.

Packet Loss attacks

DOCS | BLOG | WEBINAR

Streaming services, such as live video or multiplayer gaming, rely on a high throughput of data. When a network becomes congested, packets may become queued, and if the queue becomes too large, packets can be lost or dropped due to capacity thresholds on your hardware. Packet Loss attacks let you replicate this condition and simulate the end user experience when a percentage of outbound network packets are unexpectedly dropped or corrupted.

Packet Loss attacks help teams validate:

- **Do we have any mechanisms in place for detecting and re-sending failed packets? What impact does this have on network saturation, latency and throughput?**
- **If we have packet corruption, does the user experience gracefully degrade, or does it fail?**
- **At what point does the system go from providing a degraded experience to failing?**
- **Once packet loss returns to a normal level, does the system and user experience recover?**

Technical use cases

Test streaming services that rely on high data throughput

Services like voice and videoconferencing, multiplayer gaming, and media streaming all rely on high-bandwidth, low latency connections. In order to maximize throughput,

broadcast and multicast services often use UDP connections instead of TCP. This allows for greater throughput and the ability to support more users, but at the cost of lower reliability and potentially dropped packets. Running this attack will demonstrate how packet loss affects the quality of the stream and the load on backend systems.

Test graceful degradation mechanisms

Graceful degradation is a way of reducing load on a system while maintaining its core functionality. For example, if a user is videoconferencing and their available bandwidth drops, the videoconferencing platform can adjust by reducing video quality, or by dropping video altogether and dedicating the remaining bandwidth to audio. Packet Loss attacks are useful for testing gradual degradation mechanisms and making sure the user experience is maintained even during an incident.

Test quality of service (QoS) policies

If a network becomes saturated with one type of traffic (e.g. streaming video), then other applications might experience higher latency or dropped packets. Quality of Service (QoS) policies can prevent this by prioritizing certain types of traffic when bandwidth becomes limited. Packet Loss attacks can simulate these saturated conditions to help you validate and fine-tune your QoS policies.

Business objectives

Prepare for peak traffic events

Packet loss and packet corruption can become more pronounced during periods of high traffic and network congestion. Packet loss attacks help teams ensure that large increases in network traffic doesn't result in data loss or data corruption.

Ensure data integrity

Depending on the application, lost or corrupted packets may result in data loss. For applications that need to comply with data regulations, any amount of data loss can cause significant problems. Running packet loss attacks lets teams validate that their systems services can reliably detect and retransmit lost packets.

Improve the user experience

Many applications are impacted by packet loss including voice and videoconferencing, multiplayer gaming, and media streaming. Using packet loss attacks, teams can better prepare their real-time systems to handle unreliable networks and improve the user experience.

Conclusion

Gremlin's various attack types cover a range of scenarios and potential failure modes. While it's natural to think of each attack as just another type of failure, it's important to think more deeply about the philosophy and best practices behind Chaos Engineering. We aren't running these attacks for the sake of creating chaos, but to verify the resilience of our systems and ensure that they behave the way we expect them to. Each attack can test a wide range of behaviors, and by using each attack effectively, we can gain a comprehensive and thorough understanding of how resilient our systems really are.

Now that you've seen how each of these attacks can be applied to your systems, consider how they can help you achieve your technical and business initiatives. Start designing experiments that make use of each attack type, and plan a time to execute them as part of a GameDay. Not only will this maximize the value you get out of Gremlin, but it will help your technology, people, and processes become more reliable.

Attack type use cases

REFERENCE TABLE

	ATTACK	USE CASES
RESOURCE	 **CPU**	• Ensure systems remain operational when CPU is starved (simulate the impact of a runaway process). • Validate monitoring and alerting configurations. • Validate scalability by triggering CPU scaling thresholds in order to make sure apps can scale quickly and effectively. • Validate load balancing/clustering by exhausting resources on a node and watching to see if traffic switches to another node. • Prepare for noisy neighbors (e.g. public clouds, shared hosting, and Kubernetes clusters). • Check configuration of resource quotas (e.g. cgroups or Kubernetes CPU limits).
	 **Memory**	• Validate that your application/system is stable when memory is low by simulating memory leaks and making sure systems can tolerate out of memory (OOM) errors. • Validate auto-scaling behavior by triggering scaling thresholds. • Proactively test memory-intensive workloads (i.e. in-memory caches and databases and machine learning). • Test performance under stress by testing the impact of swapping/paging on app performance and/or testing how the app performs with noisy neighbors. • Check the configuration of resources quotas (e.g. cgroups and Kubernetes pod limits). • Validate monitoring and alerting configurations
	 **Disk**	• Test how your system handles unexpected spikes in data throughput. • Test whether your applications keep working as disks fill. • Monitor the impact of system performance when disk space is low. • Confirm that you have enough capacity reserved for a large data set (e.g. a restored backup, replicated database, or runaway log file). • Test automatic disk cleanup / compression processes (e.g. log rotation). • Test dynamic storage provisioning (e.g. sharding). • Validate limits enforced by disk quota systems, like cgroups and Kubernetes volumes. • Validate monitoring and alerting configurations.
	 **I/O**	• Test the impact of disk I/O on latency by simulating a disk-heavy workload. • Test how increasing I/O operations impact CPU and/or RAM usage. • Simulate high-latency storage devices (like network storage) to prepare for a migration from on-premises data storage to cloud storage. • Validate the effectiveness of a cache (e.g. a separate dedicated cache like Redis, or a delayed write cache). • Validate the effectiveness of faster storage devices (e.g. SSD or NVMe). • Measure IOPS (input/output operations per second) of a storage device. • Simulate an unavailable disk (e.g. Amazon EBS outage).

cont.

Attack type use cases

REFERENCE TABLE *cont.*

	ATTACK	USE CASES
STATE	**Process killer**	• Confirm that watchdog processes (like Systemd) work as intended by monitoring and restarting failed processes. • Test leader re-election for clustered workloads (e.g. Kafka and Kubernetes). • Terminate a container to see if your container orchestrator (e.g. Kubernetes) automatically restarts it.
	Shutdown	• Validate high availability of clusters (e.g. Kafka and Kubernetes) by making sure that clusters keep running without having an inconsistent state or "split brain", and/or validating that workloads are properly migrated/replicated to other nodes. • Test that failed nodes are auto-restarted or auto-replaced by your platform. • Validate that load balancers route traffic away from failed nodes.
	Time travel	• Prepare for Daylight Savings Time (DST). • Ensure security by testing TLS certificate expiration and/or determining the effect of clock skew on encryption. • Prepare for "end of epoch" problems, like Y2K and 2038. • Test clock sync (NTP) between nodes by testing for data replication when file timestamps don't match (e.g. MongoDB).

NETWORK	**Blackhole**	• Dependency Failure Testing
		  ○ - Determine which dependencies are critical and which are non-critical.
		  ○ - Validate error-handling mechanisms: failover, error messages, retry mechanisms.
		  ○ - Test for graceful degradation: does the app fail gracefully, or crash?
		• Testing reliability of clusters
		  ○ - Test for High Availability (HA)/recoverability by isolating nodes.
		  ○ - Prepare for split brain scenarios.
		• Ensure that load balancers are configured properly.
		• Ensure that monitoring and alerting are set up properly.

	DNS	• Prepare for DNS provider outages • Test mechanisms to failover to a secondary provider. • Ensure traffic gets rerouted back to the primary provider when it's back online.
	Latency	• Test application performance under saturated and unstable network conditions. • Prepare to migrate workloads from on-prem to cloud and/or clustered environments. • Test load balancing: does traffic get routed away from slow nodes to faster ones? • Fine-tune retry and timeout thresholds. • See how your application behaves over poor Internet connections. • Test concurrency control.
	Packet Loss	• Test streaming services that rely on high data throughput (voice/video conferencing, multi-player gaming, media streaming, etc.). • Test quality of service (QoS) policies. • Test and validate gradual degradation (e.g. reducing audio/video quality if the user has a poor connection)

Attack type technical and business initiatives

REFERENCE TABLE

	ATTACK	TECHNICAL INITATIVES	BUSINESS INITATIVES
RESOURCE	**CPU**	• Stress testing • Validate autoscaling • Validate monitoring and alerting • Validate resource quotas • Right-size infrastructure	• Prepare for peak traffic events • Streamline cloud migrations • Cost optimization
	Memory	• Prepare for memory leaks and memory intensive workloads • Right-size infrastructure • Validate autoscaling • Prepare for a cloud migration	• Prepare for peak traffic events • Streamline cloud migrations • Cost optimization
	Disk	• Right-size infrastructure • Test automatic processes and provisioning • Validate monitoring	• Cost optimization
	I/O	• Right-size infrastructure • Test disk-heavy workloads • Validate caches	• Prepare for peak traffic events • Improve UX • Streamline product launches
STATE	**Process killer**	• Validate redundancy and failover • Validate cluster integrity	• Maximize uptime
	Shutdown	• Validate replication and failover • Validate cluster availability and integrity • Validate automatic restarts	• Maximize uptime
	Time travel	• Prepare for time-based events like DST, leap years, and “end of epch” problems • Test TLS certificate expiration • Test clock sync between systems	• Security and compliance
NETWORK	**Blackhole**	• Test dependency failures • Validate cluster availability and load balancing • Validate monitoring and alerting • Test redundancy and failover	• Prepare for peak traffic • Maximize uptime • Business continuity and disaster recovery planning • Streamline cloud migrations
	DNS	• Prepare for DNS outages	• Maximize uptime
	Latency	• Streamline cloud migrations • Optimize network performance • Validate load balancing • Validate timeout and retry thresholds • Test concurrency control	• Prepare for peak traffic events • Improve UX • Streamline cloud migrations
	Packet Loss	• Test data throughput and integrity • Test graceful degradation • Validate quality of service (QoS)	• Prepare for peak traffic events • Meet data integrity and compliance requirements • Improve UX

Technical questions and corresponding attack type

REFERENCE TABLE

	ATTACK	QUESTIONS THIS ATTACK HELPS ANSWER
RESOURCE	**CPU**	• Do I have cleanup scripts to get rid of corrupted threads? • Is monitoring properly in place to detect CPU spikes? • What is the user experience like when CPU resources are exhausted?
	Memory	• How does my application degrade as memory resources become scarce? • Do we have alerting set up to detect when memory becomes scarce? Is it working? • When the application degrades due to memory loss, is user data lost?
	Disk	• Do we have alerts set up to tell us when the disk is close to capacity? Is the lead time long enough? • When the disk is almost full, do we know how to mitigate the situation? • Is user data lost when the disk is full and the application fails?
	I/O	• What happens when the application can't write content as expected? Does the user have to upload content again? • How much latency can the system handle before the content becomes corrupted? • Are you alerted as I/O becomes a bottleneck in performance and/or function?
STATE	**Process killer**	• If a key process dies, such as httpd in a web server, what happens to your system? Is your website entirely unavailable? • Do you have recovery mechanisms and if so, do they detect and restart the killed process? • How much user traffic is affected and for how long when the process running your application (like Tomcat) dies?
	Shutdown	• Does my session continue on a new instance gracefully? • Do enough new instances start up to handle new requests? • Is there any data loss or are ongoing processing jobs restarted? • Validate that load balancers route traffic away from failed nodes.
	Time travel	• How does my application behave if the clock skews by seconds? By minutes? • Do we have an automated process that can handle expiring certificates? • Is any data lost or corrupted when our master and read replica databases don't have the same timestamp?
NETWORK	**Blackhole**	• Where do dependencies exist within my system? • Do you have monitoring in place to alert on the unavailability of each service? • Does your application gracefully degrade? • Is the user experience negatively affected when a downstream dependency is unavailable? • Do you have non-critical dependencies that bring down the whole system?
	DNS	• Do you have a secondary DNS server and do your services fallback automatically to it? • Do you gracefully reroute calls back to the primary once it's back online? • How do your services behave during a greater DNS service outage, such as DynDNS?
	Latency	• Do you have enough replication configured to handle bursty network traffic? • Is the load being redistributed correctly to prevent data loss? • Is the user experience negatively affected when messages are delayed? • What's the recovery time to return to a real time experience?
	Packet Loss	• Does the user experience degrade gracefully? • At what point does the system fail vs. offering a degraded experience? • Does the system and user experience recover once packet loss has stopped?

Gremlin

Gremlin is the enterprise Chaos Engineering platform on a mission to help build a more reliable internet. Their solutions turn failure into resilience by offering engineers a fully hosted SaaS platform to safely experiment on complex systems, in order to identify weaknesses before they impact customers and cause revenue loss.

ADDITIONAL RESOURCES

- Attacks documentation
- Blog series
- Webinar series