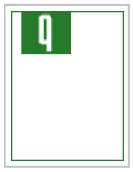
Practice
Vol. 12 No. 7 – July 2014



The Network is Reliable
An informal survey of real-world communications failures

"The network is reliable" tops Peter Deutsch's classic list, "Eight fallacies of distributed computing" (https://blogs.oracle.com/jag/resource/Fallacies.html), "all [of which] prove to be false in the long run and all [of which] cause big trouble and painful learning experiences." Accounting for and understanding the implications of network behavior is key to designing robust distributed programs; in fact, six of Deutsch's "fallacies" directly pertain to limitations on networked communications. This should be unsurprising: the ability (and often requirement) to communicate over a shared channel is a defining characteristic of distributed programs, and many of the key results in the field pertain to the possibility and impossibility of performing distributed computations under particular sets of network conditions.

by Peter Bailis, Kyle Kingsbury

Privacy, Anonymity, and Big Data in the Social Sciences
Quality social science research and the privacy of human subjects requires trust.

Open data has tremendous potential for science, but, in human subjects research, there is a tension between privacy and releasing high-quality open data. Federal law governing student privacy and the release of student records suggests that anonymizing student data protects student privacy. Guided by this standard, we de-identified and released a data set from 16 MOOCs (massive open online courses) from MITx and HarvardX on the edX platform. In this article, we show that these and other de-identification procedures necessitate changes to data sets that threaten replication and extension of baseline analyses. To balance student privacy and the benefits of open data, we suggest focusing on protecting privacy without anonymizing data by instead expanding policies that compel researchers to uphold the privacy of the subjects in open data sets. If we want to have high-quality social science research and also protect the privacy of human subjects, we must eventually have trust in researchers. Otherwise, we'll always have the strict tradeoff between anonymity and science illustrated here.

by Jon P. Daries, Justin Reich, Jim Waldo, Elise M. Young, Jonathan Whittinghill, Daniel Thomas Seaton, Andrew Dean Ho, Isaac Chuang

Securing the Tangled Web
Preventing script injection vulnerabilities through software design

Script injection vulnerabilities are a bane of Web application development: deceptively simple in cause and remedy, they are nevertheless surprisingly difficult to prevent in large-scale Web development.

by Christoph Kern

acmqueue

The Network is Reliable

An informal survey of real-world communications failures

Peter Bailis, UC Berkeley
Kyle Kingsbury, Jepsen Networks

"The network is reliable" tops Peter Deutsch's classic list, "Eight fallacies of distributed computing" (https://blogs.oracle.com/jag/resource/Fallacies.html), "all [of which] prove to be false in the long run and all [of which] cause big trouble and painful learning experiences." Accounting for and understanding the implications of network behavior is key to designing robust distributed programs—in fact, six of Deutsch's "fallacies" directly pertain to limitations on networked communications. This should be unsurprising: the ability (and often requirement) to communicate over a shared channel is a defining characteristic of distributed programs, and many of the key results in the field pertain to the possibility and impossibility of performing distributed computations under particular sets of network conditions.

For example, the celebrated FLP impossibility result[9] demonstrates the inability to guarantee consensus in an asynchronous network (i.e., one facing indefinite communication partitions between processes) with one faulty process. This means that, in the presence of unreliable (untimely) message delivery, basic operations such as modifying the set of machines in a cluster (i.e., maintaining group membership, as systems such as Zookeeper are tasked with today) are not guaranteed to complete in the event of both network asynchrony and individual server failures. Related results describe the inability to guarantee the progress of serializable transactions,[7] linearizable reads/writes,[11] and a variety of useful, programmer-friendly guarantees (http://www.bailis.org/papers/hat-vldb2014.pdf) under adverse conditions.[3] The implications of these results are not simply academic: these impossibility results have motivated a proliferation of systems and designs offering a range of alternative guarantees in the event of network failures.[5] Under a friendlier, more reliable network that guarantees timely message delivery, however, FLP and many of these related results no longer hold:[8] by making stronger guarantees about network behavior, we can circumvent the programmability implications of these impossibility proofs.

Therefore, the degree of reliability in deployment environments is critical in robust systems design and directly determines the kinds of operations that systems can reliably perform without waiting. Unfortunately, the degree to which networks are *actually* reliable in the real world is the subject of considerable and evolving debate. Some people have claimed that networks are reliable (or that partitions are rare enough in practice) and that we are too concerned with designing for theoretical failure modes. Conversely, others attest that partitions do occur in their deployments, and that, as James Hamilton of AWS (Amazon Web Services) neatly summarizes (http://perspectives.mvdirona.com/2010/04/07/StonebrakerOnCAPTheoremAndDatabases.aspx), "Network partitions should be rare but net gear continues to cause more issues than it should." So who's right?

A key challenge in this discussion is the lack of evidence. We have few normalized bases for comparing network and application reliability—and even less data. We can track link availability

and estimate packet loss, but understanding the end-to-end effect on applications is more difficult. The scant evidence we have is difficult to generalize: it is often deployment-specific and closely tied to particular vendors, topologies, and application designs. Worse, even when organizations have a clear picture of their network's behavior, they rarely share specifics. Finally, distributed systems are designed to resist failure, which means that *noticeable* outages often depend on complex interactions of failure modes. Many applications silently degrade when the network fails, and resulting problems may not be understood for some time, if ever.

As a result, much of what we believe about the failure modes of real-world distributed systems is founded on guesswork and rumor. Sysadmins and developers will swap stories over beer, but detailed, public postmortems and comprehensive surveys of network availability are few and far between. In this article, we'd like to informally bring a few of these stories (which, in most cases, are unabashedly anecdotal) together. Our focus is on descriptions of actual network behavior when possible and (more often), when not, on the implications of network failures and asynchrony for real-world systems deployments. We believe this is a first step toward a more open and honest discussion of real-world partition behavior, and, ultimately, toward more robust distributed systems design.

RUMBLINGS FROM LARGE DEPLOYMENTS

To start off, let's consider evidence from big players in distributed systems: companies running globally distributed infrastructure with hundreds of thousands of servers. These reports perhaps best summarize operations in the large, distilling the experience of operating what are likely the biggest distributed systems ever deployed. These companies' publications (unlike many of the reports we will examine later) often capture aggregate system behavior and large-scale statistical trends, and indicate (often obliquely) that partitions are of concern in their deployments.

THE MICROSOFT DATA-CENTER STUDY

A team from the University of Toronto and Microsoft Research studied the behavior of network failures in several of Microsoft's data centers.[12] They found an average failure rate of 5.2 devices per day and 40.8 links per day, with a median time to repair of approximately five minutes (and a maximum of one week). While the researchers note that correlating link failures and communication partitions is challenging, they estimate a median packet loss of 59,000 packets per failure. Perhaps of more concern is their finding that network redundancy improves median traffic by only 43 percent—that is, network redundancy does not eliminate common causes of network failure.

HP ENTERPRISE MANAGED NETWORKS

A joint study between researchers at the University of California, San Diego, and HP Labs examined the causes and severity of network failures in HP's managed networks by analyzing support-ticket data (http://www.hpl.hp.com/techreports/2012/HPL-2012-101.pdf). "Connectivity"-related tickets accounted for 11.4 percent of support tickets (14 percent of which were of the highest-priority level), with a median incident duration of 2 hours and 45 minutes for the highest-priority tickets and a median duration of 4 hours and 18 minutes for all tickets.

GOOGLE CHUBBY

Google's paper (http://research.google.com/archive/chubby-osdi06.pdf) describing the design and operation of Chubby, its distributed lock manager, outlines the root causes of 61 outages over 700 days of operation across several clusters. Of the nine outages that lasted more than 30 seconds, four were caused by network maintenance and two were caused by "suspected network connectivity problems."

GOOGLE'S DESIGN LESSONS FROM DISTRIBUTED SYSTEMS

In Design Lessons and Advice from Building Large Scale Distributed Systems (http://www.cs.cornell.edu/projects/ladis2009/talks/dean-keynote-ladis2009.pdf), Google Fellow Jeff Dean suggested that a typical first year for a new Google cluster involves:

- Five racks going wonky (40-80 machines seeing 50 percent packet loss).
- Eight network maintenance events (four of which might cause ~30-minute random connectivity losses).
- Three router failures (resulting in the need to pull traffic immediately for an hour).

While Google doesn't tell us much about the application-level consequences of its network partitions, Dean suggested that they were of concern, citing the perennial challenge of creating "easy-to-use abstractions for resolving conflicting updates to multiple versions of a piece of state," useful for "reconciling replicated state in different data centers after repairing a network partition."

AMAZON DYNAMO

Amazon's Dynamo paper (http://www.allthingsdistributed.com/files/amazon-dynamo-sosp2007.pdf) frequently cites the incidence of partitions as a key design consideration. Specifically, the authors note that they rejected designs from "traditional replicated relational database systems" because they "are not capable of handling network partitions."

YAHOO! PNUTS/SHERPA

Yahoo! PNUTS/Sherpa was designed as a distributed database operating in geographically distinct data centers. Originally, PNUTS supported a strongly consistent "timeline consistency" operation, with one master per data item. The developers noted, however, that in the event of network partitioning or server failures, this design decision was too restrictive for many applications:[16]

> The first deployment of Sherpa supported the timeline-consistency model—namely, all replicas of a record apply all updates in the same order—and has API-level features to enable applications to cope with asynchronous replication. Strict adherence leads to difficult situations under network partitioning or server failures. These can be partially addressed with override procedures and local data replication, but in many circumstances, applications need a relaxed approach.

According to the same report, PNUTS now offers weaker consistency alternatives providing availability during partitions.

DATA-CENTER NETWORK FAILURES

Data-center networks are subject to power failure, misconfiguration, firmware bugs, topology changes, cable damage, and malicious traffic. Their failure modes are accordingly diverse.

POWER FAILURE ON BOTH REDUNDANT SWITCHES

As Microsoft's SIGCOMM paper suggests, redundancy doesn't always prevent link failure. When a power distribution unit failed and took down one of two redundant top-of-rack switches, Fog Creek lost service for a subset of customers on that rack but remained consistent and available for most users. The other switch in that rack, however, *also* lost power for undetermined reasons. That failure isolated the two neighboring racks from each other, taking down all On Demand services.

SWITCH SPLIT-BRAIN CAUSED BY BPDU FLOOD

During a planned network reconfiguration to improve reliability, Fog Creek Software suddenly lost access to its network.[10]

> A network loop had formed between several switches. The gateways controlling access to the switch management network were isolated from each other, generating a split-brain scenario. Neither was accessible due to a...multi-switch BPDU (bridge protocol data unit) flood, indicating a spanning-tree flap. This is most likely what was changing the loop domain.

According to the BPDU standard, the flood shouldn't have happened. But it did, and this deviation from the system's assumptions resulted in two hours of total service unavailability.

BRIDGE LOOPS, MISCONFIGURATION, BROKEN MAC CACHES

To address high latencies caused by a daisy-chained network topology, Github installed a set of aggregation switches in its data center (https://github.com/blog/1346-network-problems-last-friday). Despite a redundant network, the installation process resulted in bridge loops, and switches disabled links to prevent failure. This problem was quickly resolved, but later investigation revealed that many interfaces were still pegged at 100 percent capacity.

While that problem was under investigation, a misconfigured switch triggered aberrant automatic fault-detection behavior: when one link was disabled, the fault detector disabled all links, leading to 18 minutes of downtime. The problem was traced to a firmware bug preventing switches from updating their MAC (media access control) address caches correctly, forcing them to broadcast most packets to every interface.

MLAG, SPANNING TREE, AND STONITH

In December 2012 (https://github.com/blog/1364-downtime-last-saturday), a planned software update on an aggregation switch caused instability at Github. To collect diagnostic information, the network vendor killed a particular software agent running on one of the aggregation switches.

Github's aggregation switches are clustered in pairs using a feature called MLAG (multi-chassis link aggregation), which presents two physical switches as a single L2 (layer-2) device. The MLAG failure-detection protocol relies on both Ethernet link state and a logical heartbeat message exchanged between nodes. When the switch agent was killed, it was unable to shut down the Ethernet link, preventing the still-healthy aggregation switch from handling link aggregation, spanning-tree, and other L2 protocols. This forced a spanning-tree leader election and reconvergence for all links, blocking all traffic between access switches for 90 seconds.

This 90-second network partition caused file servers using Pacemaker and DRBD (Distributed

Replicated Block Device) for HA (high availability) failover to declare each other dead, and to issue STONITH (shoot the other node in the head) messages to one another. The network partition delayed delivery of those messages, causing some file-server pairs to believe they were *both* active. When the network recovered, both nodes shot each other at the same time. With both nodes dead, files belonging to the pair were unavailable.

To prevent file-system corruption, DRBD requires that administrators ensure the original primary node is still the primary node before resuming replication. For pairs where both nodes were primary, the ops team had to examine log files or bring each node online in isolation to determine its state. Recovering those downed file-server pairs took five hours, during which Github service was significantly degraded.

CLOUD NETWORKS

Large-scale virtualized environments are notorious for transient latency, dropped packets, and full-blown network partitions, often affecting a particular software version or availability zone. Sometimes the failures occur between specific subsections of the provider's data center, revealing planes of cleavage in the underlying hardware topology.

AN ISOLATED MONGODB PRIMARY ON EC2

In a comment on Call me maybe: MongoDB (http://aphyr.com/posts/284-call-me-maybe-mongodb), Scott Bessler observed exactly the same failure mode Kyle demonstrated earlier:

> [This scenario] happened to us today when EC2 West region had network issues that caused a network partition that separated PRIMARY from its 2 SECONDARIES in a 3 node replset. 2 hours later the old primary rejoined and rolled back everything on the new primary.

This partition caused two hours of write loss. From our conversations with large-scale MongoDB users, we gather that network events causing failover on Amazon's EC2 (Elastic Compute Cloud) are common. Simultaneous primaries accepting writes for multiple days are anecdotally common.

MNESIA SPLIT-BRAIN ON EC2

Outages can leave two nodes connected to the Internet but unable to see each other. This type of partition is especially dangerous, as writes to both sides of a partitioned cluster can cause inconsistency and lost data. Paul Mineiro reports exactly this scenario in an Mnesia cluster (http://dukesoferl.blogspot.com/2008/03/network-partition-oops.html?m=1), which diverged overnight. The cluster's state wasn't critical, so the operations team simply nuked one side of the cluster. They conclude: "The experience has convinced us that we need to prioritize up our network partition recovery strategy."

EC2 INSTABILITY CAUSING MONGODB AND ELASTICSEARCH UNAVAILABILITY

Network disruptions in EC2 can affect only certain groups of nodes. For example, one report of a total partition between the front-end and back-end servers (https://forums.aws.amazon.com/thread.jspa?messageID=454155) states that a site's servers lose their connections to all back-end instances for a few seconds, several times a month. Even though the disruptions were short, they resulted in

30- to 45-minute outages and a corrupted index for ElasticSearch. As problems escalated, the outages occurred "2 to 4 times a day."

AWS EBS OUTAGE

On April 21, 2011, AWS suffered unavailability for 12 hours,[2] causing hundreds of high-profile Web sites to go offline. As a part of normal AWS scaling activities, Amazon engineers had shifted traffic away from a router in the EBS (Elastic Block Store) network in a single U.S. East AZ (Availability Zone), but, due to incorrect routing policies:

> ...many EBS nodes in the affected Availability Zone were completely isolated from other EBS nodes in its cluster. Unlike a normal network interruption, this change disconnected both the primary and secondary network simultaneously, leaving the affected nodes completely isolated from one another.

The partition, coupled with aggressive failure-recovery code, caused a mirroring storm that caused network congestion and triggered a previously unknown race condition in EBS. EC2 was unavailable for roughly 12 hours, and EBS was unavailable or degraded for more than 80 hours.

The EBS failure also caused an outage in Amazon's RDS (Relational Database Service). When one AZ fails, RDS is designed to failover to a different AZ; however, 2.5 percent of multi-AZ databases in U.S. East failed to failover because of a bug in the failover protocol.

This correlated failure caused widespread outages for clients relying on AWS. For example, Heroku reported between 16 and 60 hours of unavailability for its users' databases.

ISOLATED REDIS PRIMARY ON EC2

On July 18, 2013, Twilio's billing system, which stores account credits in Redis, failed.[19] A network partition isolated the Redis primary from all secondaries. Because Twilio did not promote a new secondary, writes to the primary remained consistent. When the primary became visible to the secondaries again, however, all secondaries simultaneously initiated a full resynchronization with the primary, overloading it and causing Redis-dependent services to fail.

The ops team restarted the Redis primary to address the high load. Upon restart, however, the Redis primary reloaded an incorrect configuration file, which caused it to enter read-only mode. With all account balances at zero, and in read-only mode, every Twilio API call caused the billing system to recharge customer credit cards automatically, resulting in 1.1 percent of customers being overbilled over a period of 40 minutes. For example, Appointment Reminder reported that every SMS message and phone call it issued resulted in a $500 charge to its credit card, which stopped accepting charges after $3,500.

Twilio recovered the Redis state from an independent billing system—a relational data store—and after some hiccups, restored proper service, including credits to affected users.

HOSTING PROVIDERS

Running your own data center can be cheaper and more reliable than using public cloud infrastructure, but it means you have to be a network and server administrator. What about hosting providers, which rent dedicated or virtualized hardware to users and often take care of the network and hardware setup for you?

AN UNDETECTED GLUSTERFS SPLIT-BRAIN

Freistil IT hosts its servers with a colocation/managed-hosting provider. Its monitoring system alerted Freistil to a 50-100 percent packet loss localized to a specific data center.[15] The network failure, caused by a router firmware bug, returned the next day. Elevated packet loss caused the GlusterFS distributed file system to enter split-brain undetected:

> ...we became aware of [problems] in the afternoon when a customer called our support hotline because their website failed to deliver certain image files. We found that this was caused by a split-brain situation...and the self-heal algorithm built into the Gluster file system was not able to resolve this inconsistency between the two data sets.

Repairing that inconsistency led to a "brief overload of the Web nodes because of a short surge in network traffic."

AN ANONYMOUS HOSTING PROVIDER

Anecdotally, many major managed hosting providers experience network failures. One company running 100-200 nodes on a major hosting provider reported that in a 90-day period the provider's network went through five distinct periods of partitions. Some partitions disabled connectivity between the provider's cloud network and the public Internet, and others separated the cloud network from the provider's internal managed-hosting network.

PACEMAKER/HEARTBEAT SPLIT-BRAIN

A post to Linux-HA details a long-running partition between a Heartbeat pair (http://readlist.com/lists/lists.linux-ha.org/linux-ha/6/31964.html), in which two Linode VMs each declared the other dead and claimed a shared IP for themselves. Successive posts suggest further network problems: e-mails failed to dispatch because of DNS (Domain Name System) resolution failure, and nodes reported, "Network unreachable." In this case, the impact appears to have been minimal, in part because the partitioned application was just a proxy.

WIDE AREA NETWORKS

While we have largely focused on failures over local area networks (or near-local networks), WAN (wide area network) failures are also common, if less frequently documented. These failures are particularly interesting because there are often fewer redundant WAN routes and because systems guaranteeing high availability (and disaster recovery) often require distribution across multiple data centers. Accordingly, graceful degradation under partitions or increased latency is especially important for geographically widespread services.

CENIC STUDY

Researchers at the UCSD analyzed five years of operation in the CENIC (Corporation for Education Network Initiatives in California) WAN,[18] which contains more than 200 routers across California. By cross-correlating link failures and additional external BGP (Border Gateway Protocol) and trace-route data, they discovered more than 500 "isolating network partitions" that caused connectivity problems between hosts. Average partition duration ranged from 6 minutes for software-related

failures to more than 8.2 hours for hardware-related failures (median 2.7 and 32 minutes; 95th percentile of 19.9 minutes and 3.7 days, respectively).

PAGERDUTY

PagerDuty designed its system to remain available in the face of node, data-center, or even provider failure; its services are replicated between two EC2 regions and a data center hosted by Linode. On April 13, 2013, an AWS peering point in northern California degraded, causing connectivity issues for one of PagerDuty's EC2 nodes. As latencies between AWS Availability Zones rose, the notification dispatch system lost quorum and stopped dispatching messages entirely.

Even though PagerDuty's infrastructure was designed with partition tolerance in mind, correlated failures caused by a shared peering point between two data centers resulted in 18 minutes of unavailability, dropping inbound API requests and delaying queued pages until quorum was reestablished.

GLOBAL ROUTING FAILURES

Despite the high level of redundancy in Internet systems, some network failures take place on a global scale.

CLOUDFLARE

CloudFlare runs 23 data centers with redundant network paths and anycast failover. In response to a DDoS (distributed denial-of-service) attack against one of its customers, the CloudFlare operations team deployed a new firewall rule to drop packets of a specific size.[17] Juniper's FlowSpec protocol propagated that rule to all CloudFlare edge routers—but then:

> What should have happened is that no packet should have matched that rule because no packet was actually that large. What happened instead is that the routers encountered the rule and then proceeded to consume all their RAM until they crashed.

Recovering from the failure was complicated by routers that failed to reboot automatically and by inaccessible management ports.

> Even though some data centers came back online initially, they fell back over again because all the traffic across our entire network hit them and overloaded their resources.

CloudFlare monitors its network carefully, and the operations team had immediate visibility into the failure. Coordinating globally distributed systems is complex, however, and calling on-site engineers to find and reboot routers by hand takes time. Recovery began after 30 minutes and was complete after an hour of unavailability.

JUNIPER ROUTING BUG

A firmware bug introduced as a part of an upgrade in Juniper Networks' routers caused outages in Level 3 Communications' networking backbone in 2011. This subsequently knocked services offline, including Time Warner Cable, RIM BlackBerry, and several UK Internet service providers.

GLOBAL BGP OUTAGES

There have been several global Internet outages related to BGP misconfiguration. Notably, in 2008 Pakistan Telecom, responding to a government edict to block YouTube.com, incorrectly advertised its (blocked) route to other providers, which hijacked traffic from the site and briefly rendered it unreachable.

In 2010 a group of Duke University researchers achieved a similar effect by testing an experimental flag in the BGP (http://www.merit.edu/mail.archives/nanog/msg11505.html). Similar incidents occurred in 2006, knocking sites such as *Martha Stewart Living* and the *New York Times* offline; in 2005, where a misconfiguration in Turkey attempted a redirect for the entire Internet; and in 1997.

NICS AND DRIVERS

Unreliable networking hardware and/or drivers are implicated in a broad array of partitions.

BCM5709 AND FRIENDS

As a classic example of NIC (network interface controller) unreliability, Marc Donges and Michael Chan (http://www.spinics.net/lists/netdev/msg210485.html) describe how their popular Broadcom BCM5709 chip dropped inbound but not outbound packets. The primary server was unable to service requests, but, because it could still *send* heartbeats to its hot spare, the spare considered the primary alive and refused to take over. Their service was unavailable for five hours and did not recover without a reboot.

Sven Ulland followed up, reporting the same symptoms with the BCM5709S chipset on Linux 2.6.32-41squeeze2. Despite pulling commits from mainline, which supposedly fixed a similar set of issues with the bnx2 driver, Ulland's team was unable to resolve the issue until version 2.6.38.

As a large number of servers shipped the BCM5709, the larger impact of these firmware bugs was widely observed. For example, the 5709 had a bug in the 802.3x flow control, leading to extraneous PAUSE frames when the chipset crashed or its buffer filled up. This problem was magnified by the BCM56314 and BCM56820 switch-on-a-chip devices (found in many top-of-rack switches), which, by default, sent PAUSE frames to any interface communicating with the offending 5709 NIC. This led to cascading failures on entire switches or networks.

The bnx2 driver could also cause transient or flapping network failures, as described in an ElasticSearch failure report. Meanwhile, the Broadcom 57711 was notorious for causing high latencies under load with jumbo frames, a particularly thorny issue for ESX users with iSCSI-backed storage.

INTEL 82574 PACKET OF DEATH

A motherboard manufacturer failed to flash the EEPROM correctly for its Intel 82574–based system. The result was a very-hard-to-diagnose error in which an inbound SIP (Session Initiation Protocol) packet of a particular structure would disable the NIC.[14] Only a cold restart would bring the system back to normal.

A GLUSTERFS PARTITION CAUSED BY A DRIVER BUG

After a scheduled upgrade, CityCloud noticed unexpected network failures in two distinct GlusterFS pairs, followed by a third.[6] Suspecting link aggregation, CityCloud disabled the feature on its switches and allowed self-healing operations to proceed.

Roughly 12 hours later, the network failures returned. CityCloud identified the cause as a driver issue and updated the downed node, returning service. The outage, however, resulted in data inconsistency between GlusterFS pairs and data corruption between virtual machine file systems.

APPLICATION-LEVEL FAILURES

Not all asynchrony originates in the physical network. Sometimes dropped or delayed messages are a consequence of crashes, program errors, operating-system scheduler latency, or overloaded processes. The following studies highlight the fact that communication failures—wherein the system delays or drops messages—can occur at any layer of the software stack, and that designs that expect synchronous communication may behave unexpectedly during periods of asynchrony.

CPU USE AND SERVICE CONTENTION

Bonsai.io discovered (http://www.bonsai.io/blog/2013/03/05/outage-post-mortem) high CPU and memory use on an ElasticSearch node combined with difficulty connecting to various cluster components, likely a consequence of an "excessively high number of expensive requests being allowed through to the cluster."

Upon restarting the servers, the cluster split into two independent components. A subsequent restart resolved the split-brain behavior, but customers complained they were unable to delete or create indices. The logs revealed that servers were repeatedly trying to recover unassigned indices, which "poisoned the cluster's attempt to service normal traffic which changes the cluster state." The failure led to 20 minutes of unavailability and six hours of degraded service.

LONG GC PAUSES AND I/O

Stop-the-world garbage collection and blocking for disk I/O can cause runtime latencies on the order of seconds to minutes. As Searchbox IO and several other production users (https://github.com/elasticsearch/elasticsearch/issues/2488) have found, GC (garbage collection) pressure in an ElasticSearch cluster can cause secondary nodes to declare a primary dead and to attempt a new election. Because of nonmajority quorum configuration, ElasticSearch elected two different primaries, leading to inconsistency and downtime. Surprisingly, even with majority quorums, due to protocol design, ElasticSearch does not currently prevent simultaneous master election; GC pauses and high IO_WAIT times due to I/O can cause split-brain behavior, write loss, and index corruption.

MYSQL OVERLOAD AND A PACEMAKER SEGFAULT

In 2012, a routine database migration caused unexpectedly high load on the MySQL primary at Github.[13] The cluster coordinator, unable to perform health checks against the busy MySQL server, decided the primary was down and promoted a secondary. The secondary had a cold cache and performed poorly, causing failover back to the original primary. The operations team manually halted this automatic failover, and the site appeared to recover.

The next morning, the operations team discovered that the standby MySQL node was no longer replicating changes from the primary. Operations decided to disable the coordinator's maintenance mode and allow the replication manager to fix the problem. Unfortunately, this triggered a segfault in the coordinator, and a conflict between manual configuration and the automated replication tools rendered github.com unavailable.

The partition caused inconsistency in the MySQL database—both between the secondary and primary, and between MySQL and other data stores such as Redis. Because foreign key relationships were not consistent, Github showed private repositories to the wrong users' dashboards and incorrectly routed some newly created repositories.

DRBD SPLIT-BRAIN

When a two-node cluster partitions, there are no cases in which a node can reliably declare itself to be the primary. When this happens to a DRBD file system, as one user reported (http://serverfault.com/questions/485545/dual-primary-ocfs2-drbd-encountered-split-brain-is-recovery-always-going-to-be), both nodes can remain online and accept writes, leading to divergent file system-level changes.

A NETWARE SPLIT-BRAIN

Short-lived failures can lead to long outages. In a Usenet post to novell.support.cluster-services, an admin reports that a two-node failover cluster running Novell NetWare experienced transient network outages. The secondary node eventually killed itself, and the primary (though still running) was no longer reachable by other hosts on the network. The post goes on to detail a series of network partition events correlated with backup jobs.

VOLTDB SPLIT-BRAIN ON EC2

One VoltDB user reports regular network failures causing replica divergence (https://forum.voltdb.com/showthread.php?552-Nodes-stop-talking-to-each-other-and-form-independent-clusters) but also indicates that the network logs included no dropped packets. Because this cluster had not enabled split-brain detection, both nodes ran as isolated primaries, causing significant data loss.

MYSTERY RABBITMQ PARTITIONS

Sometimes, nobody knows why a system partitions. This RabbitMQ failure (http://serverfault.com/questions/497308/rabbitmq-network-partition-error) seems like one of those cases: few retransmits, no large gaps between messages, and no clear loss of connectivity between nodes. Increasing the partition-detection time-out to two minutes reduced the frequency of partitions but didn't prevent them altogether.

ELASTICSEARCH DISCOVERY FAILURE ON EC2

Another EC2 split-brain (http://elasticsearch-users.115913.n3.nabble.com/EC2-discovery-leads-to-two-masters-td3239318.html): a two-node cluster failed to converge on "roughly 1 out of 10 startups" when discovery messages took longer than three seconds to exchange. As a result, both nodes would start as primaries with the same cluster name. Since ElasticSearch doesn't demote primaries automatically, split-brain persisted until administrators intervened. Increasing the discovery time-out to 15 seconds resolved the issue.

RABBITMQ AND ELASTICSEARCH ON WINDOWS AZURE

There are a few scattered reports of Windows Azure partitions, such as one account (http://rabbitmq.1065348.n5.nabble.com/Instable-HA-cluster-td24690.html) of a RabbitMQ cluster that entered split-brain on a weekly basis. There's also a report of an ElasticSearch split-brain (https://

groups.google.com/forum/?fromgroups#!topic/elasticsearch/muZtKij3nUw), but since Azure is a relative newcomer compared with EC2, descriptions of its network reliability are limited.

CONCLUSIONS: WHERE DO WE GO FROM HERE?

This article is meant as a reference point—to illustrate that, according to a wide range of (often informal) accounts, communication failures occur in many real-world environments. Processes, servers, NICs, switches, and local and wide area networks can all fail, with real economic consequences. Network outages can suddenly occur in systems that have been stable for months at a time, during routine upgrades, or as a result of emergency maintenance. The consequences of these outages range from increased latency and temporary unavailability to inconsistency, corruption, and data loss. Split-brain is not an academic concern: it happens to all kinds of systems—sometimes for days on end. Partitions deserve serious consideration.

On the other hand, some networks really are reliable. Engineers at major financial firms have anecdotally reported that despite putting serious effort into designing systems that gracefully tolerate partitions, their networks rarely, if ever, exhibit partition behavior. Cautious engineering and aggressive network advances (along with lots of money) can prevent outages. Moreover, in this article, we have presented failure scenarios; we acknowledge it's much harder to demonstrate that network failures have *not* occurred.

Not all organizations, however, can afford the cost or operational complexity of highly reliable networks. From Google and Amazon (which operate commodity and/or low-cost hardware because of sheer scale) to one-person startups built on shoestring budgets, communication-isolating network failures are a real risk, in addition to the variety of other failure modes (including human error) that real-world distributed systems face.

It's important to consider this risk before a partition occurs, because it's much easier to make decisions about partition behavior on a whiteboard than to redesign, reengineer, and upgrade a complex system in a production environment—especially when it's throwing errors at your users. For some applications, failure is an option—but you should characterize and explicitly account for it as a part of your design. Finally, given the additional latency[1] and coordination benefits[4] of partition-aware designs, you might just find that accounting for these partitions delivers benefits in the average case as well.

We invite you to contribute your own experiences with or without network partitions. Open a pull request on https://github.com/aphyr/partitions-post (which, incidentally, contains all references), leave a comment, write a blog post, or release a postmortem. Data will inform this conversation, future designs, and, ultimately, the availability of the systems on which we all depend.

REFERENCES

1. Abadi, D. 2012. Consistency tradeoffs in modern distributed database system design: CAP is only part of the story. *Computer* 45(2): 37-42; http://dl.acm.org/citation.cfm?id=2360959.
2. Amazon Web Services. 2011. Summary of the Amazon EC2 and Amazon RDS service disruption in the U.S. East region; http://aws.amazon.com/message/65648/.
3. Bailis, P., Davidson, A., Fekete, A., Ghodsi, A., Hellerstein, J.M., Stoica, I. 2014. Highly available transactions: virtues and limitations. In *Proceedings of VLDB* (to appear); http://www.bailis.org/papers/hat-vldb2014.pdf.

4. Bailis, P., Fekete, A., Franklin, M.J., Ghodsi, A., Hellerstein, J.M., Stoica, I. 2014. Coordination-avoiding database systems; http://arxiv.org/abs/1402.2237.
5. Bailis, P., Ghodsi, A. 2013. Eventual consistency today: limitations, extensions, and beyond. *ACM Queue* 11(3); http://queue.acm.org/detail.cfm?id=2462076.
6. CityCloud. 2011; https://www.citycloud.eu/cloud-computing/post-mortem/.
7. Davidson, S.B., Garcia-Molina, H., Skeen, D. 1985. Consistency in a partitioned network: a survey. *ACM Computing Surveys* 17(3): 341-370; http://dl.acm.org/citation.cfm?id=5508.
8. Dwork, C., Lynch, M., Stockmeyer, L. 1988. Consensus in the presence of partial synchrony. *Journal of the ACM* 35(2): 288-323. http://dl.acm.org/citation.cfm?id=42283.
9. Fischer, M. J., Lynch, N.A., Patterson, M.S. 1985. Impossibility of distributed consensus with one faulty process. *Journal of the ACM* 32(2): 374-382; http://dl.acm.org/citation.cfm?id=214121.
10. Fog Creek Software. 2012. May 5-6 network maintenance post-mortem; http://status.fogcreek.com/2012/05/may-5-6-network-maintenance-post-mortem.html.
11. Gilbert, S., Lynch, N. 2002. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant Web services. *ACM SIGACT News* 33(2): 51-59; http://dl.acm.org/citation.cfm?id=564601.
12. Gill, P., Jain, N., Nagappan, N. 2011. Understanding network failures in data centers: measurement, analysis, and implications. In *Proceedings of SIGCOMM*; http://research.microsoft.com/en-us/um/people/navendu/papers/sigcomm11netwiser.pdf.
13. Github. 2012. Github availability this week; https://github.com/blog/1261-github-availability-this-week.
14. Kielhofner, K. 2013. Packets of death; http://blog.krisk.org/2013/02/packets-of-death.html.
15. Lillich, J. 2013. Post mortem: network issues last week; http://www.freistil.it/2013/02/post-mortem-network-issues-last-week/.
16. Narayan, P.P.S. 2010. Sherpa update; https://developer.yahoo.com/blogs/ydn/sherpa-7992.html#4.
17. Prince, M. 2013. Today's outage post mortem; http://blog.cloudflare.com/todays-outage-post-mortem-82515.
18. Turner, D., Levchenko, K., Snoeren, A., Savage, S. 2010. California fault lines: understanding the causes and impact of network failures. In *Proceedings of SIGCOMM* ; http://cseweb.ucsd.edu/~snoeren/papers/cenic-sigcomm10.pdf.
19. Twilio. 2013. Billing incident post-mortem: breakdown, analysis and root cause; http://www.twilio.com/blog/2013/07/billing-incident-post-mortem.html.

LOVE IT, HATE IT? LET US KNOW

feedback@queue.acm.org

PETER BAILIS is a graduate student of computer science and a member of the AMPLab and BOOM projects at UC Berkeley. He currently studies database and distributed systems, with a particular focus on fast and scalable data serving and transaction processing. He holds an A.B. from Harvard College and is the recipient of the NSF Graduate Research Fellowship and the Berkeley Fellowship for Graduate Study. He blogs regularly at http://bailis.org/blog and tweets as @pbailis.

KYLE KINGSBURY is the author of Riemann, Timelike, and a slew of other open-source packages. In his free time he verifies distributed systems' safety claims as a part of the Jepsen project.

acmqueue

Privacy, Anonymity, and Big Data in the Social Sciences

Quality social science research and the privacy of human subjects requires trust

Jon P. Daries

Open data has tremendous potential for science, but, in human subjects research, there is a tension between privacy and releasing high-quality open data. Federal law governing student privacy and the release of student records suggests that anonymizing student data protects student privacy. Guided by this standard, we de-identified and released a data set from 16 MOOCs (massive open online courses) from MITx and HarvardX on the edX platform. In this article, we show that these and other de-identification procedures necessitate changes to data sets that threaten replication and extension of baseline analyses. To balance student privacy and the benefits of open data, we suggest focusing on protecting privacy *without* anonymizing data by instead expanding policies that compel researchers to uphold the privacy of the subjects in open data sets. If we want to have high-quality social science research and also protect the privacy of human subjects, we must eventually have trust in researchers. Otherwise, we'll always have the strict tradeoff between anonymity and science illustrated here.

The *open* in massive open online course has many interpretations. Some MOOCs are hosted on open-source platforms, some use only openly licensed content, and most MOOCs are openly accessible to any learner without fee or prerequisites. We would like to add one more notion of openness: open access to data generated by MOOCs. We argue that this is part of the responsibility of MOOCs, and that fulfilling this responsibility threatens current conventions of anonymity in policy and public perception.

In this spirit of open data, on May 30, 2014, a team of researchers from Harvard and MIT (including this author team) announced the release of an open data set containing student records from 16 courses conducted in the first year of the edX platform. (In May 2012, MIT and Harvard launched edX, a nonprofit platform for hosting and marketing MOOCs. MITx and HarvardX are the two respective institutional organizations focused on MOOCs.)[6] The data set is a de-identified version of that used to publish *HarvardX and MITx: The First Year of Open Online Courses,* a report revealing findings about student demographics, course-taking patterns, certification rates, and other measures of student behavior.[6] The goal for this data release was twofold: first, to allow other researchers to replicate the results of the analysis; and second, to allow researchers to conduct novel analyses beyond the original work, adding to the body of literature about open online courses.

Within hours of the release, original analysis of the data began appearing on Twitter, with figures and source code. Two weeks after the release, the data journalism team at *The Chronicle of Higher Education* published "8 Things You Should Know about MOOCs," an article that explored new dimensions of the data set, including the gender balance of the courses.[13] Within the first month of the release, the data had been downloaded more than 650 times. With surprising speed, the data set began fulfilling its purpose: to allow the research community to use open data from online learning platforms to advance scientific progress.

The rapid spread of new research from this data is exciting, but the excitement is tempered by a necessary limitation of the released data: it represents a subset of the complete data. To comply with federal regulations on student privacy, the released data set had to be de-identified. This article demonstrates tradeoffs between the need to meet the demands of federal regulations of student privacy, on the one hand, and our responsibility to release data for replication and downstream analyses, on the other. For example, the original analysis found that approximately 5 percent of course registrants earned certificates. Some methods of de-identification cut that percentage in half.

It is impossible to anonymize identifiable data without the possibility of affecting some future analysis in some way. It is possible to quantify the difference between replications from the de-identified data and original findings; however, it is difficult to fully anticipate whether findings from novel analyses will result in valid insights or artifacts of de-identification. Higher standards for de-identification can lead to lower-value de-identified data. This could have a chilling effect on the motivations of social science researchers. If findings are likely to be biased by the de-identification process, why should researchers spend their scarce time on de-identified data?

At the launch of edX in May 2012, the presidents of MIT and Harvard spoke about the edX platform, and the data generated by it, as a public good. If academic and independent researchers alike have access to data from MOOCs, then the progress of research into online education will be faster and results can be furthered, refined, and tested. These ideals for open MOOC data are undermined, however, if protecting student privacy means that open data sets are markedly different from the original data. The tension between privacy and open data is in need of a better solution than anonymized data sets. Indeed, the fundamental problem in our current regulatory framework may be an unfortunate and unnecessary conflation of privacy and anonymity. Jeffrey Skopek[17] of Harvard Law School outlines the difference between the two as follows:

> under the condition of privacy, we have knowledge of a person's identity, but not of an associated personal fact, whereas under the condition of anonymity, we have knowledge of a personal fact, but not of the associated person's identity. In this sense, privacy and anonymity are flip sides of each other. And for this reason, they can often function in opposite ways: whereas privacy often hides facts about someone whose identity is known by removing information and other goods associated with the person from public circulation, anonymity often hides the identity of someone about whom facts are known for the purpose of putting such goods into public circulation (p. 1755).

Realizing the potential of open data in social science requires a new paradigm for the protection of student privacy: either a technological solution such as differential privacy,[3] which separates analysis from possession of the data, or a policy-based solution that allows open access to possibly re-identifiable data while policing the uses of the data.

This article describes the motivations behind efforts to release learner data, the contemporary regulatory framework of student privacy, our efforts to comply with those regulations in creating an open data set from MOOCs, and some analytical consequences of de-identification. From this case study in de-identification, we conclude that the scientific ideals of open data and the current regulatory requirements concerning anonymizing data are incompatible. Resolving that

incompatibility will require new approaches that better balance the protection of privacy and the advancement of science in educational research and the social sciences more broadly.

BALANCING OPEN DATA AND STUDENT PRIVACY REGULATIONS

As with open-source code and openly licensed content, support for open data has been steadily building. In the United States, government agencies have increased their expectations for sharing research data.[5] In 2003 the National Institutes of Health became the first federal agency to require research grant applicants to describe their plans for data sharing.[12] In 2013 the Office of Science and Technology Policy released a memorandum requiring the public storage of digital data from unclassified, federally funded research.[7] These trends dovetailed with growing interest in data sharing in the learning sciences community. In 2006 researchers from Carnegie Mellon University opened DataShop, a repository of event logs from intelligent tutoring systems and one of the largest sources of open data in educational research outside the federal government.[8]

Open data has tremendous potential across the scientific disciplines to facilitate greater transparency through replication and faster innovation through novel analyses. It is particularly important in research into open, online learning such as MOOCs. A study released earlier this year[1] estimates that more than 7 million people in the United States alone have taken at least one online course, and that that number is growing by 6 percent each year. These students are taking online courses at a variety of institutions, from community colleges to research universities, and open MOOC data will facilitate research that could be helpful to all institutions with online offerings.

Open data can also facilitate cooperation between researchers with different domains of expertise. As George Siemens, president of the Society for Learning Analytics Research, has argued, learning research involving large and complex data sets requires interdisciplinary collaboration between data scientists and educational researchers.[16] Open data sets make it easier for researchers in these two distinct domains to come together.

While open educational data has great promise for advancing science, it also raises important questions about student privacy. In higher education, the cornerstone of student privacy law is FERPA (Family Educational Rights and Privacy Act). FERPA is a federal privacy statute that regulates access to and disclosure of a student's educational records. In our de-identification procedures, we aimed to comply with FERPA, although not all institutions consider MOOC learners to be subject to FERPA.[11]

FERPA offers protections for PII (personally identifiable information) within student records. Per FERPA, PII cannot be disclosed, but if PII is removed from a record, then the student becomes anonymous, privacy is protected, and the resulting de-identified data can be disclosed to anyone (20 U.S.C. § 1232g(b)(1) 2012; 34 C.F.R. § 99.31(b) 2013). FERPA thus equates anonymity the removal of PII—with privacy.

FERPA's PII definition includes some statutorily defined categories, such as name, address, social security number, and mother's maiden name, but also

> other information that, alone or in combination, is linked or linkable to a specific student that would allow a reasonable person in the school community, who does not have personal knowledge of the relevant circumstances, to identify the student with reasonable certainty (34 C.F.R. § 99.3, 2013).

In assessing the reasonable certainty of identification, the educational institution is supposed to take into account other data releases that might increase the chance of identification.[22] Therefore, an adequate de-identification procedure must remove not only statutorily required elements, but also quasi-identifiers. These quasi-identifiers are pieces of information that can be uniquely identifying in combination with each other or with additional data sources from outside the student records. They are not defined by statute or regulatory guidance from the Department of Education but left up to the educational institution to define.[22]

The potential for combining quasi-identifiers to uniquely identify individuals is well established. For example, Latanya Sweeney,[21] from the School of Computer Science at Carnegie Mellon University, has demonstrated that 87 percent of the U.S. population can be uniquely identified with a reasonable degree of certainty by a combination of ZIP code, date of birth, and gender. These risks are further heightened in open, online learning environments because of the public nature of the activity. As another example, some MOOC students participate in course discussion forums—which, for many courses, remain available online beyond the course end date. Students' usernames are displayed beside their posts, allowing for linkages of information across courses, potentially revealing students who enroll for unique combinations of courses. A very common use of the discussion forums early in a course is a self-introduction thread where students state their age and location, among other PII.

Meanwhile, another source of identifying data is social media. It is conceivable that students could verbosely log their online education on Facebook or Twitter, tweeting as soon as they register for a new course or mentioning their course grade in a Facebook post. Given these external sources, an argument can be made that many columns in the person-course data set that would not typically be thought of as identifiers could qualify as quasi-identifiers.

The regulatory framework defined by FERPA guided our efforts to de-identify the person-course data set for an open release. Removing direct identifiers such as students' usernames and IP addresses was straightforward, but the challenge of dealing with quasi-identifiers was more complicated. We opted for a framework of *k*-anonymity.[20] A data set is *k*-anonymous if any one individual in the data set cannot be distinguished from at least *k*-1 other individuals in the same data set. This requires ensuring that no individual has a combination of quasi-identifiers different from *k*-1 others. If a data set cannot meet these requirements, then the data must be modified to meet *k*-anonymity, either by generalizing data within cases or suppressing entire cases. For example, if a single student in the data set is from Latvia, we can employ one of these remedies: generalize her location by reporting her as from Europe rather than Latvia, for example; suppress her location information; or suppress her case entirely.

This begins to illustrate the fundamental tension between generating data sets that meet the requirements of anonymity mandates and advancing the science of learning through public releases of data. Protecting student privacy under the current regulatory regime requires modifying data to ensure that individual students cannot be identified. These modifications can, however, change the data set considerably, raising serious questions about the utility of the open data for replication or novel analysis. The next sections describe our approach to generating a *k*-anonymous data set, and then examine the consequences of our modifications to the size and nature of the data set.

DE-IDENTIFICATION METHODS

The original, identified person-course data set contained the following information:

- Information about students (username, IP address, country, self-reported level of education, self-reported year of birth, and self-reported gender).
- The course ID (a string identifying the institution, semester, and course).
- Information about student activity in the course (date and time of first interaction, date and time of last interaction, number of days active, number of chapters viewed, number of events recorded by the edX platform, number of video play events, number of forum posts, and final course grade).
- Four variables computed to indicate level of course involvement (*registered*: enrolled in the course; *viewed*: interacted with the courseware at least once; *explored*: interacted with content from more than 50 percent of course chapters; and *certified*: earned a passing grade and received a certificate).

Transforming this person-course data set into a *k*-anonymous data set that we believed met FERPA guidelines required four steps: 1) defining identifiers and quasi-identifiers; 2) defining the value for *k*; 3) removing identifiers; and 4) modifying or deleting values of quasi-identifiers from the data set in a way that ensures *k*-anonymity, while minimizing changes to the data set.

We defined two variables in the original data set as identifiers and six variables as quasi-identifiers. The username was considered identifying in and of itself, so we replaced it with a random ID. IP address was also removed. Four student demographic variables were defined as quasi-identifiers: country, gender, age, and level of education. Course ID was considered a quasi-identifier since students might take unique combinations of courses and because it provides a link between PII posted in forums and the person-course data set. The number of forum posts made by a student was also a quasi-identifier because a determined individual could scrape the content of the forums from the archived courses and then identify users with unique numbers of forum posts.

Once the quasi-identifiers were chosen, we had to determine a value of *k* to use for implementing *k*-anonymity. In general, larger values of *k* require greater changes to de-identify, and smaller values of *k* leave data sets more vulnerable to re-identification. The U.S. Department of Education offers guidance to the de-identification process in a variety of contexts, but it does not recommend or require specific values of *k* for specific contexts. In one FAQ, the department's Privacy Technical Assistance Center states that many "statisticians consider a cell size of 3 to be the absolute minimum" and goes on to say that values of 5 to 10 are even safer.[15] We chose a *k* of 5 for our de-identification.

Since our data set contained registrations for 16 courses, registrations in multiple courses could be used for re-identification. The *k*-anonymity approach would ensure that no individual was uniquely identifiable using the quasi-identifiers *within* a course, but further care had to be taken to remove the possibility that a registrant could be uniquely identified based upon registering in a unique combination or number of courses. For example, if only three people registered for all 16 courses, then those three registrants would not be *k*-anonymous across courses, and some of their registration records would need to be suppressed in order to lower the risk of their re-identification.

The key part of the de-identification process was modifying the data such that no combination of quasi-identifiers described groups consisting of fewer than five students. The two tools employed for this task were *generalization,* the combining of more granular values into categories (e.g., 1, 2, 3, 4, and 5 become "1-5"); and *suppression,* the deletion of data that compromises *k*-anonymity.[21] Many strategies for de-identification, including Sweeney s Datafly algorithm, implement both tools with

different amounts of emphasis on one technique or the other.[18] More generalization would mean that fewer records are suppressed, but the remaining records would be less specific than the original data. A heavier reliance on suppression would remove more records from the data, but the remaining records would be less altered.

The following section illustrates differential tradeoffs between valid research inferences and de-identification methods by comparing two de-identification approaches: one that favors generalization over suppression (hereafter referred to as the generalization emphasis, or GE, method), and one that favors suppression over generalization (hereafter referred to as the suppression emphasis, or SE, method). There are other ways of approaching the problem of de-identification, but these were two that were easily implemented. Our intent is not to discern the dominance of one technique over the other in any general case but rather to show that tradeoffs between anonymity and valid research inferences a) are unavoidable and b) will depend on the method of de-identification.

The SE method used generalization for the names of countries (grouping them into continent/region names for countries with fewer than 5,000 rows) and for the first- and last-event time stamps (grouping them into dates by truncating the hour and minute portion of the time stamps). Suppression was then employed for rows that were not *k*-anonymous across the quasi-identifying variables. For more information on the specifics of the implementation, please refer to the documentation accompanying the data release.[10]

The GE method generalized year of birth into groups of two (e.g., 1980–1981), and number of forum posts into groups of five for values greater than 10 (e.g., 11–15). Suppression was then employed for rows that were not *k*-anonymous across the quasi-identifying variables. The generalizations resulted in a data set that needed less suppression than in the SE method, but also reduced the precision of the generalized variables.

Both de-identification processes are more likely to suppress registrants in smaller courses: the smaller a course, the higher the chances that any given combination of demographics would not be *k*-anonymous, and the more likely that this row would need to be suppressed. Furthermore, since an activity variable (number of forum posts) was included as a quasi-identifier, both methods were likely to remove users who were more active in the forums. Since only 8 percent of students had any posts in the forums at all, and since these students were typically active in other ways, the records of many of the most active students were suppressed.

THE CONSEQUENCES OF TWO APPROACHES TO DE-IDENTIFICATION

Both of the de-identified data sets differ from the original data set in substantial ways. We reproduced analyses conducted on the original data set and evaluated the magnitude of changes in the new data sets. This section highlights those differences.

Both de-identified data sets are substantially smaller than the original data set (see table 1), but de-identification did not affect enrollment numbers uniformly across courses. Table 1 shows the percentage decrease of enrollment in each de-identified data set compared with the original file. Only a small percentage of records from CS50x were removed because CS50x was hosted off the edX platform; thus, we have no data about forum usage (one of our quasi-identifying variables).

Table 2 shows that de-identification has a disproportionate impact on the most active students.

TABLE 1 **Percent decrease in records by course and by de-identification method**

Institution	Course Code	Baseline N	GE Reduction	SE Reduction	Average Reduction
HarvardX	CS50x	181,410	4%	6%	5%
MITx	6.002x	51,394	15%	21%	18%
MITx	6.00x	72,920	15%	21%	18%
MITx	6.00x	84,511	16%	21%	18%
MITx	6.002x	29,050	17%	23%	20%
MITx	8.02x	41,037	17%	24%	21%
HarvardX	PH278x	53,335	18%	26%	22%
HarvardX	ER22x	79,750	21%	28%	25%
MITx	14.73x	39,759	22%	30%	26%
HarvardX	CB22x	43,555	23%	31%	27%
HarvardX	PH207x	61,170	25%	32%	28%
MITx	3.091x	24,493	33%	42%	37%
MITx	8.MReV	16,787	33%	44%	38%
MITx	7.00x	37,997	35%	45%	40%
MITx	3.091x	12,276	39%	50%	44%
MITx	2.01x	12,243	44%	54%	49%
Total		841,687	18%	24%	21%

Andrew Dean Ho et al.[6] identified four mutually exclusive categories of students: *Only Registered* enrolled in the course but did not interact with the courseware; *Only Viewed* interacted with at least one, and fewer than half, of the course chapters; *Only Explored* interacted with content from half or more of the course chapters but did not earn a certificate; and *Certified* earned a certificate in the course. Table 2 shows that the proportions of students in each category seem to change only slightly after de-identification; however, the percentage of certified students in the de-identified data set is nearly half the percentage in the original data set. Given the policy concerns around MOOC certification rates, this is a substantially important difference, even if only a small change in percentage points.

Demographic data from the de-identified data sets was similar to the original person-course data set. Table 3 shows the distributions of gender and bachelor's degree attainment, respectively, for each data set. The proportions of bachelor's degree holders in all three data sets are nearly identical. The

TABLE 2 **Percent decrease in records by activity category and by de-identification method**

Activity Category	Baseline N	Baseline Percentage	GE Percentage	SE Percentage	GE Change	SE Change
Only Registered	292,852	34.8%	37.3%	37.6%	+2.5%	+2.8%
Only Viewed	469,702	55.8%	56.2%	56.1%	+0.4%	+0.3%
Only Explored	35,937	4.3%	3.6%	3.5%	-0.7%	-0.7%
Certified	43,196	5.1%	2.9%	2.8%	-2.2%	-2.4%
Total	841,687	100%	100%	100%		

TABLE 3 **Changes in demographics and activity by de-identification method**

Statistic	Baseline	GE	SE	GE % Change	SE % Change
Percent Bachelor's or Higher	63%	63%	63%	0.1%	-0.2%
Percent Female	29%	26%	26%	-2.2%	-2.9%
Median Number of Events (explored + certified)	3645	2194	2052	-40%	-44%

de-identified data sets report slightly lower percentages of female students than the original data set. The gender bias of MOOCs is a sensitive policy issue, so this difference raises concerns about analyses conducted with the de-identified data sets.

The suppression of highly active users substantially reduces the median number of total events in the courseware. Table 3 shows the median events for all three data sets, and the de-identified data sets have median event values that are two-thirds of the value reported by the original data set.

Finally, we analyzed the correlations among variables in all three of the data sets. We use correlations to illustrate possible changes in predictive models that rely on correlation and covariance matrices, from the regression-based prediction of grades to principal components analyses and other multivariate methods. Although straight changes in correlations are dependent on base rates, and averages of correlations are not well formed, we present these simple statistics here for ease of interpretation. No correlation changed direction, and all remained significant at the 0.05 level. For all registrants, the SE data set reported correlations marginally closer to the original data set than the GE method, while for explored and certified students only, the GE data set was slightly closer to the original (see table 4).

It is possible to use the results from the previous tables to formulate a multivariate model that has population parameters in these tables. By generating data from such a model in proportion to the numbers we have in the baseline data set, we would enable researchers to replicate the correlations and mean values above. Such a model, however, would lead to distorted results for any analysis that is not implied by the multivariate model selected. In addition, the unusual distributions seen in MOOC data[2] would be difficult to model using conventional distributional forms.

The comparisons presented here between the de-identified data sets and the original data set provide evidence for the tension between protecting anonymity and releasing useful data. We emphasize that the differences identified here are not those that may be most concerning. The above analyses characterize the differences that researchers conducting replication studies might expect to see. For novel analyses that have yet to be performed on the data, it is difficult to formulate an a priori estimate of the impact of de-identification. For researchers hoping to use de-identified, public data sets to advance research, this means that any given finding might be the result of perturbations from de-identification.

BETTER OPTIONS FOR SCIENCE AND PRIVACY WITH RESPECT TO MOOC DATA

As illustrated in the previous section, the differences between the de-identified data set and the original data range from small changes in the proportion of various demographic categories to large

TABLE 4 **Changes in Pearson Correlations by de-identification method and activity category**

Variable 1	Variable 2	Registrants	Baseline Correlation	GE Correlation	SE Correlation	GE Change (+/-)	SE Change (+/-)
Grade	Number of days active	All	0.800	0.750	0.745	-0.050	-0.055
Grade	Number of days active	Explored + Certified	0.553	0.558	0.564	+0.005	+0.011
Grade	Number of events	All	0.722	0.701	0.697	-0.021	-0.025
Grade	Number of events	Explored + Certified	0.458	0.495	0.501	+0.037	+0.043
Grade	Number of forum posts	All	0.146	0.064	0.156	-0.082	+0.010
Grade	Number of forum posts	Explored + Certified	0.074	0.036	0.108	-0.038	+0.034
Grade	Number of video plays	All	0.396	0.397	0.403	+0.001	+0.007
Grade	Number of video plays	Explored + Certified	0.159	0.194	0.189	+0.035	+0.030
Number of events	Number of days active	All	0.844	0.837	0.835	-0.007	-0.009
Number of events	Number of days active	Explored + Certified	0.736	0.773	0.776	+0.037	+0.040
Number of events	Number of video plays	All	0.665	0.698	0.714	+0.033	+0.049
Number of events	Number of video plays	Explored + Certified	0.587	0.628	0.634	+0.041	+0.047
Number of forum posts	Number of days active	All	0.207	0.104	0.207	-0.103	+0.000
Number of forum posts	Number of days active	Explored + Certified	0.180	0.103	0.200	-0.077	+0.020
Number of forum posts	Number of events	All	0.287	0.117	0.194	-0.170	-0.093
Number of forum posts	Number of events	Explored + Certified	0.279	0.113	0.176	-0.166	-0.103
Number of forum posts	Number of video plays	All	0.091	0.035	0.100	-0.056	+0.009
Number of forum posts	Number of video plays	Explored + Certified	0.051	0.014	0.050	-0.037	-0.001
Number of video plays	Number of days active	All	0.474	0.492	0.505	+0.018	+0.031
Number of video plays	Number of days active	Explored + Certified	0.311	0.404	0.407	+0.093	+0.096
Average		All	0.463	0.420	0.456	-0.044	-0.008
Average		Explored + Certified	0.339	0.332	0.361	-0.007	+0.022

decreases in activity variables and certification rates. It is quite possible that analyses not yet thought of would yield even more dramatic differences between the two data sets. Even if a de-identification method is found that maintains many of the observed research results from the original data set, there can be no guarantee that other analyses will not have been corrupted by de-identification.

At this point it may be possible to take for granted that any standard for de-identification will increase over time. Information is becoming more accessible, and researchers are increasingly sophisticated and creative about possible re-identification strategies. Cynthia Dwork of Microsoft Research, in a presentation at the Big Data Privacy Workshop sponsored by MIT and the White House in early 2014, pointed out that de-identification efforts have been progressing as a sort of arms race, similar to advances in the field of cryptography.[4] Although *k*-anonymity is a useful heuristic, researchers have challenged that it alone is not sufficient. Ashwin Machanavajjhala et al.[9] point out that a *k*-anonymous data set is still vulnerable to a "homogeneity attack." If, after undergoing a process that ensures *k*-anonymity, there exists a group of size *k* or larger for whom the value of a sensitive variable is homogenous (i.e., all members of the group have the same value), then the value of that sensitive variable is effectively disclosed even if the attacker does not know exactly

which record belongs to the target. Machanavajjhala et al. define this principle as *l*-diversity. Other researchers have advanced an alphabet soup of critiques to *k*-anonymity such as *m*-invariance and *t*-similarity.[4] Even if it were possible to devise a de-identification method that did not impact statistical analysis, it could quickly become outmoded by advances in re-identification techniques.

This example of our efforts to de-identify a simple set of student data—a tiny fraction of the granular event logs available from the edX platform—reveals a conflict between open data, the replicability of results, and the potential for novel analyses on one hand, and the anonymity of research subjects on the other. This tension extends beyond MOOC data to much of social science data, but the challenge is acute in educational research because FERPA conflates anonymity and therefore de-identification—with privacy. One conclusion could be that the data is too sensitive to share; so if de-identification has too large an impact on the integrity of a data set, then the data should not be shared. We believe that this is an undesirable position, because the few researchers privileged enough to have access to the data would then be working in a bubble where few of their peers have the ability to challenge or augment their findings. Such limits would, at best, slow down the advancement of knowledge. At worst, these limits would prevent groundbreaking research from ever being conducted.

Neither abandoning open data nor loosening student privacy protections is a wise option. Rather, the research community should vigorously pursue technology and policy solutions to the tension between open data and privacy. A promising technological solution is differential privacy.[3] Under the framework of differential privacy, the original data is maintained, but raw PII is not accessed by the researcher. Instead, it resides in a secure database that has the ability to answer questions about the data. A researcher can submit a model—a regression equation, for example—to the database, and the regression coefficients and R-squared are returned. Differential privacy has challenges of its own, and remains an open research question because implementing such a system would require carefully crafting limits around the number and specificity of questions that can be asked in order to prevent identification of subjects. For example, no answer could be returned if it drew upon fewer than *k* rows, where *k* is the same minimum cell size used in *k*-anonymity.

Policy changes may be more feasible in the short term. An approach suggested by the U.S. PCAST (President's Council of Advisors on Science and Technology) is to accept that anonymization is an obsolete tactic made increasingly difficult by advances in data mining and big data.[14] PCAST recommends that privacy policy emphasize that the use of data should not compromise privacy and should focus on the what rather than the how.[14] One can imagine a system whereby researchers accessing an open data set would agree to use the data only to pursue particular ends, such as research, and not to contact subjects for commercial purposes or to rerelease the data. Such a policy would need to be accompanied by provisions for enforcement and audits, and the creation of practicable systems for enforcement is, admittedly, no small feat.

We propose that privacy can be upheld by researchers bound to an ethical and legal framework, *even if* these researchers can identify individuals and all of their actions. If we want to have high-quality social science research and privacy of human subjects, we must eventually have trust in researchers. Otherwise, we'll always have a strict tradeoff between anonymity and science.

REFERENCES

1. Allen, I. E., Seaman, J. 2014. Grade change: tracking online education in the United States; http://sloanconsortium.org/publications/survey/grade-change-2013.

2. DeBoer, J., Ho, A. D., Stump, G. S., Breslow, L. 2013. Changing course: reconceptualizing educational variables for massive open online courses. *Educational Researcher*. Published online before print February 7, 2014.
3. Dwork, C. 2006. Differential privacy. *Automata, Languages and Programming*. Springer Berlin Heidelberg: 1–12.
4. Dwork, C. 2014. State of the art of privacy protection; video http://web.mit.edu/bigdata-priv/agenda.html.
5. Goben, A. Salo, D. 2013. Federal research: data requirements set to change. *College & Research Libraries News* 74(8): 421425; http://crln.acrl.org/content/74/8/421.full.
6. Ho, A. D., Reich, J., Nesterko, S., Seaton, D. T., Mullaney, T., Waldo, J., Chuang, I. 2014. HarvardX and MITx: the first year of open online courses, fall 2012-summer 2013; http://ssrn.com/abstract=2381263.
7. Holdren, J. P. 2013. Increasing access to the results of federally funded scientific research; http://www.whitehouse.gov/sites/default/files/microsites/ostp/ostp_public_access_memo_2013.pdf.
8. Koedinger, K. R., Baker, R. S. J. d., Cunningham, K., Skogsholm, A., Leber, B., Stamper, J. 2010. A data repository for the EDM community: The PSLC DataShop. In *Handbook of Educational Data Mining*, ed. C. Romero, S. Ventura, M. Pechenizkiy, R. S. J. d. Baker. Boca Raton, FL: CRC Press.
9. Machanavajjhala, A., Gehrke, J., Kifer, D., Venkitasubramaniam, M. 2007. L-diversity: privacy beyond k-anonymity. *ACM Transactions on Knowledge Discovery from Data (TKDD)* 1(1): 3.
10. MITx and HarvardX. 2014. HarvardX-MITx person-course academic year 2013 de-identified dataset, version 2.0; http://dx.doi.org/10.7910/DVN/26147.
11. MOOCs @ Illinois. 2013. *FAQ for Faculty*; http://mooc.illinois.edu/resources/faqfaculty/.
12. National Institutes of Health. 2003. Final NIH statement on sharing research data; http://grants.nih.gov/grants/guide/notice-files/NOT-OD-03-032.html.
13. Newman, J. Oh, S. 2014. 8 things you should know about MOOCs. *The Chronicle of Higher Education* (June 13); http://chronicle.com/article/8-Things-You-Should-Know-About/146901/.
14. President's Council of Advisors on Science and Technology. 2014. Big data and privacy: a technological perspective; http://www.whitehouse.gov/sites/default/files/microsites/ostp/PCAST/pcast_big_data_and_privacy_-_may_2014.pdf.
15. Privacy Technical Assistance Center. 2012. Frequently asked questions—disclosure avoidance; http://ptac.ed.gov/sites/default/files/FAQs_disclosure_avoidance.pdf.
16. Siemens, G. 2014. The Journal of Learning Analytics: supporting and promoting learning analytics research. *Journal of Learning Analytics* 1(1): 3–5; http://epress.lib.uts.edu.au/journals/index.php/JLA/article/view/3908/4010.
17. Skopek, J. M. 2014. Anonymity, the production of goods, and institutional design. *Fordham Law Review* 82(4): 1751–1809; http://ir.lawnet.fordham.edu/flr/vol82/iss4/4/.
18. Sweeney, L. 1998. Datafly: a system for providing anonymity in medical data. In *Database Security, XI: Status and Prospects*, ed. T. Lin, and S. Qian. Amsterdam: Elsevier Science.
19. Sweeney, L. 2000. Simple demographics often identify people uniquely. *Health (San Francisco)* 671: 1–34.
20. Sweeney, L. 2002a. *k*-anonymity: a model for protecting privacy. *International Journal on Uncertainty, Fuzziness and Knowledge-based Systems* 10(5): 557–570.
21. Sweeney, L. (2002b). Achieving *k*-anonymity privacy protection using generalization and

suppression. *International Journal on Uncertainty, Fuzziness and Knowledge-based Systems* 10(5): 571–588.
22. United States Department of Education. 2008. Family educational rights and privacy. *Federal Register* 73(237). Washington, DC: U.S. Government Printing Office; http://www.gpo.gov/fdsys/pkg/FR-2008-12-09/pdf/E8-28864.pdf.

LOVE IT, HATE IT? LET US KNOW
feedback@queue.acm.org

The authors are a group of researchers and administrators from MIT and Harvard who have been working with the data, and policies related to the data, from the MITx and HarvardX MOOCs on the edX platform:
JON P. DARIES, Massachusetts Institute of Technology
JUSTIN REICH, Harvard University
JIM WALDO, Harvard University
ELISE M. YOUNG, Harvard University
JONATHAN WHITTINGHILL, Harvard University
DANIEL THOMAS SEATON, Massachusetts Institute of Technology
ANDREW DEAN HO, Harvard University
ISAAC CHUANG, Massachusetts Institute of Technology

acmqueue

Securing the Tangled Web

Preventing script injection vulnerabilities through software design

Christoph Kern, Google

Script injection vulnerabilities are a bane of Web application development: deceptively simple in cause and remedy, they are nevertheless surprisingly difficult to prevent in large-scale Web development.

XSS (cross-site scripting)[2,7,8] arises when insufficient data validation, sanitization, or escaping within a Web application allow an attacker to cause browser-side execution of malicious JavaScript in the application's context. This injected code can then do whatever the attacker wants, using the privileges of the victim. Exploitation of XSS bugs results in complete (though not necessarily persistent) compromise of the victim's session with the vulnerable application. This article provides an overview of how XSS vulnerabilities arise and why it is so difficult to avoid them in real-world Web application software development. The article then describes software design patterns developed at Google to address the problem. A key goal of these design patterns is to confine the potential for XSS bugs to a small fraction of an application's code base, significantly improving one's ability to reason about the absence of this class of security bugs. In several software projects within Google, this approach has resulted in a substantial reduction in the incidence of XSS vulnerabilities.

XSS VULNERABILITIES

Most commonly, XSS vulnerabilities result from insufficiently validating, sanitizing, or escaping strings that are derived from an *untrusted source* and passed along to a *sink* that interprets them in a way that may result in script execution.

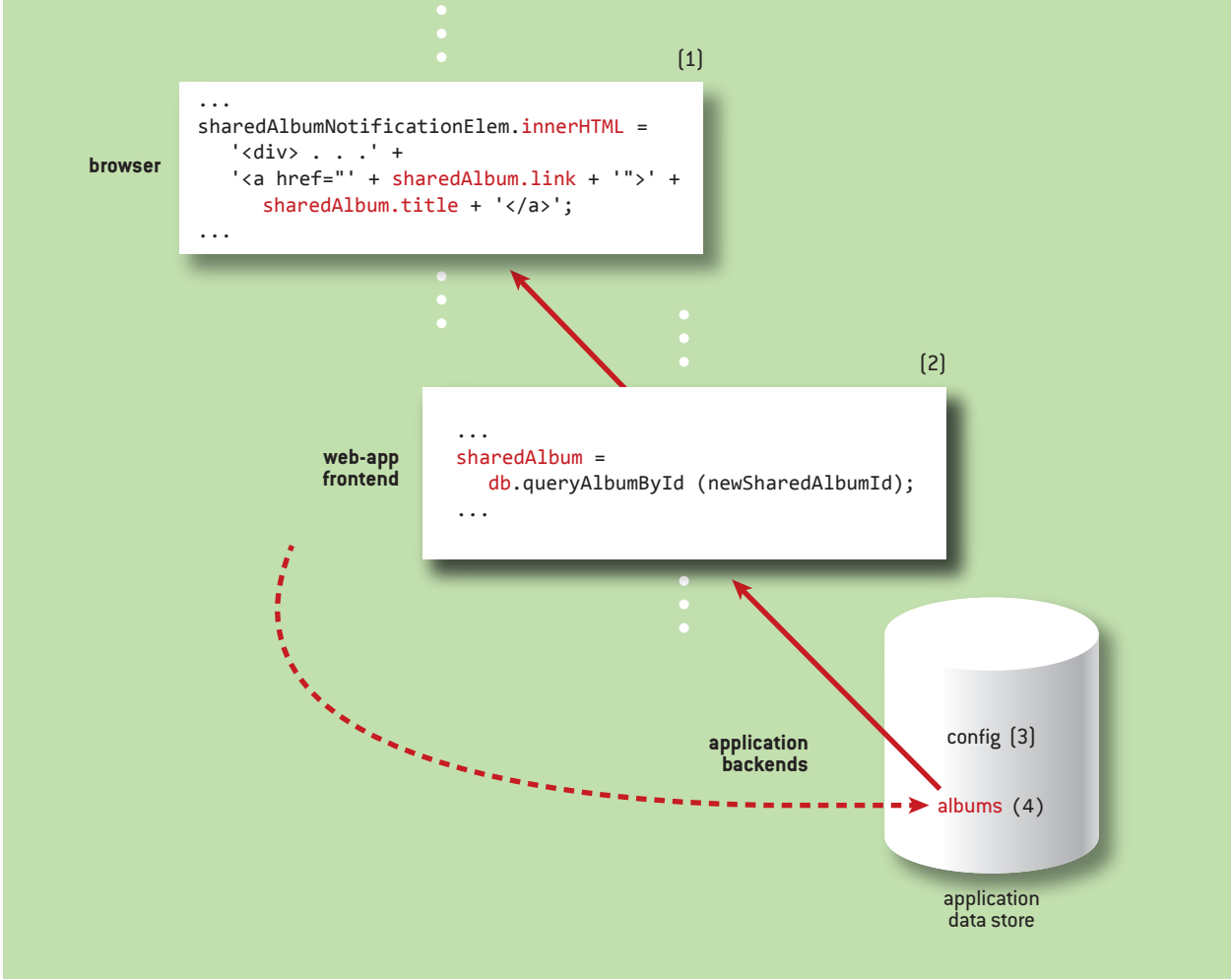
Common sources of untrustworthy data include HTTP request parameters, as well as user-controlled data located in persistent data stores. Strings are often concatenated with or interpolated into larger strings before assignment to a sink. The most frequently encountered sinks relevant to XSS vulnerabilities are those that interpret the assigned value as HTML markup, which includes server-side HTTP responses of MIME-type `text/html`, and the `Element.prototype.innerHTML` DOM (Document Object Model)[7] property in browser-side JavaScript code.

Figure 1a shows a slice of vulnerable code from a hypothetical photo-sharing application. Like many modern Web applications, much of its user-interface logic is implemented in browser-side JavaScript code, but the observations made in this article transfer readily to applications whose UI is implemented via traditional server-side HTML rendering.

In code snippet (1) in the figure, the application generates HTML markup for a notification to be shown to a user when another user invites the first user to view a photo album. The generated markup is assigned to the `innerHTML` property of a DOM element (a node in the hierarchical object representation of UI elements in a browser window), resulting in its evaluation and rendering.

The notification contains the album's title, chosen by the *second* user. A malicious user can create an album titled:

FIGURE 1A **Vulnerable Code of a Hypothetical Photo-Sharing Application**



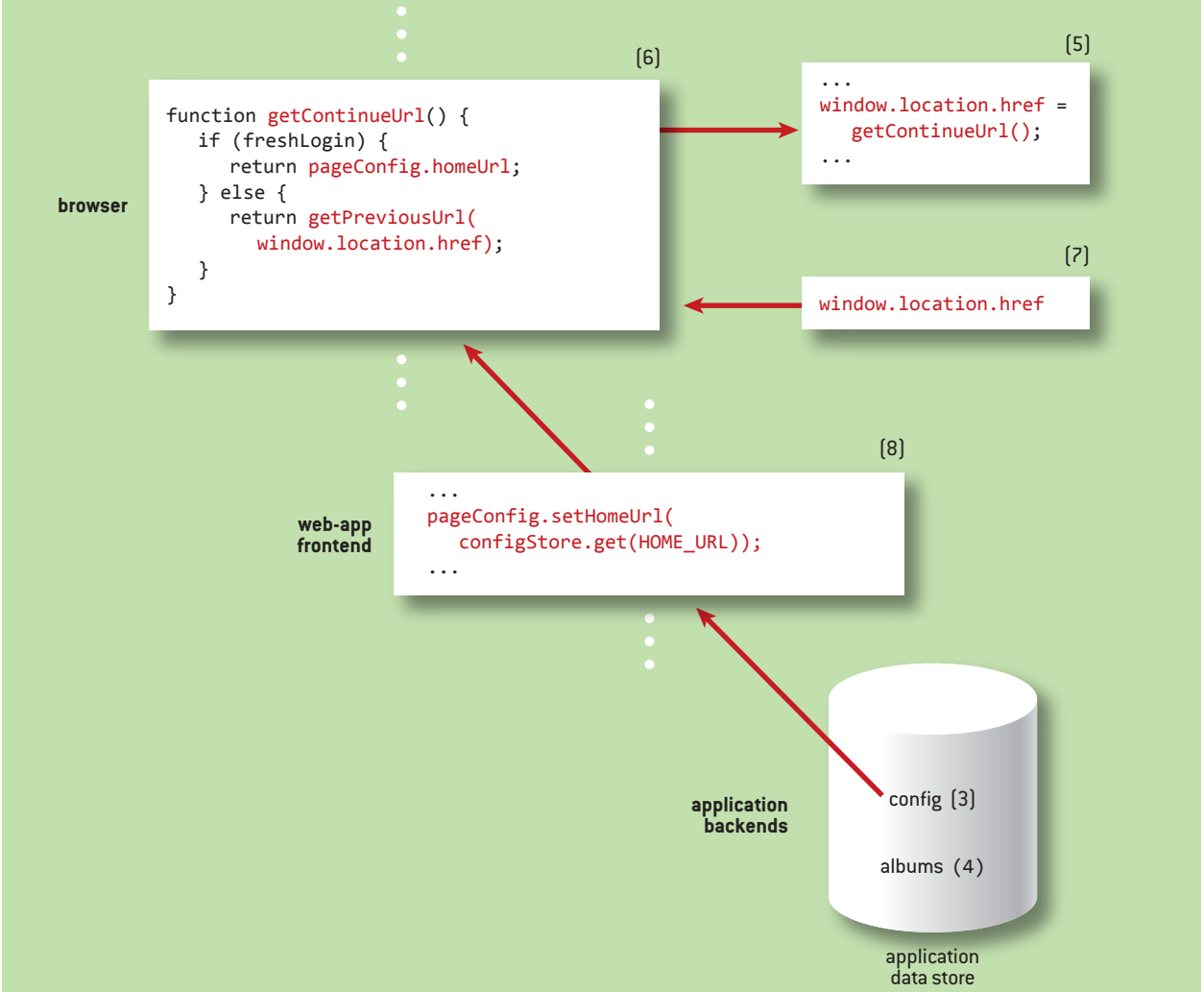

```
<script>attackScript;</script>
```

Since no escaping or validation is applied, this attacker-chosen HTML is interpolated as-is into the markup generated in code snippet (1). This markup is assigned to the `innerHTML` sink, and hence evaluated in the context of the victim's session, executing the attacker-chosen JavaScript code.

To fix this bug, the album's title must be *HTML-escape*d before use in markup, ensuring that it is interpreted as plain text, not markup. HTML-escaping replaces HTML metacharacters such as <, >, ", ', and & with corresponding character entity references or numeric character references: `<`, `>`, `&quot;`, `&#39;`, and `&`. The result will then be parsed as a substring in a text node or attribute value and will not introduce element or attribute boundaries.

As noted, most data flows with a potential for XSS are into sinks that interpret data as HTML markup. But other types of sinks can result in XSS bugs as well: figure 1b shows another slice of the previously mentioned photo-sharing application, responsible for navigating the user interface after

FIGURE 1B **Another Slice of the Photo-Sharing Application**



a login operation. After a fresh login, the app navigates to a preconfigured URL for the application's main page. If the login resulted from a session time-out, however, the app navigates back to the URL the user had visited before the time-out. Using a common technique for short-term state storage in Web applications, this URL is encoded in a parameter of the current URL.

The page navigation is implemented via assignment to the `window.location.href` DOM property, which browsers interpret as an instruction to navigate the current window to the provided URL. Unfortunately, navigating a browser to a URL of the form `javascript:attackScript` causes execution of the URL's body as JavaScript. In this scenario, the target URL is extracted from a parameter of the *current* URL, which is generally under attacker control (a malicious page visited by a victim can instruct the browser to navigate to an attacker-chosen URL).

Thus, this code is also vulnerable to XSS. To fix the bug, it is necessary to *validate* that the URL

will not result in script execution when dereferenced, by ensuring that its scheme is benign—for example, `https`.

WHY IS XSS SO DIFFICULT TO AVOID?

Avoiding the introduction of XSS into nontrivial applications is a difficult problem in practice: XSS remains among the top vulnerabilities in Web applications, according to OWASP (Open Web Application Security Project);[4] within Google it is the most common class of Web application vulnerabilities among those reported under Google's Vulnerability Reward Program (https://www. google.com/about/appsecurity/reward-program/).

Traditionally, advice (including my own) on how to prevent XSS has largely focused on:

- Training developers to treat (by sanitization, validation, and/or escaping) untrustworthy values interpolated into HTML markup.[2,5]
- Security-reviewing and/or testing code for adherence to such guidance.

In our experience at Google, this approach certainly helps reduce the incidence of XSS, but for even moderately complex Web applications, it does not prevent introduction of XSS to a reasonably high degree of confidence. We see a combination of factors leading to this situation.

SUBTLE SECURITY CONSIDERATIONS

As seen, the requirements for secure handling of an untrustworthy value depend on the context in which the value is used. The most commonly encountered context is string interpolation within the content of HTML markup elements; here, simple HTML-escaping suffices to prevent XSS bugs. Several special contexts, however, apply to various DOM elements and within certain kinds of markup, where embedded strings are interpreted as URLs, CSS (Cascading Style Sheets) expressions, or JavaScript code. To avoid XSS bugs, each of these contexts requires specific validation or escaping, or a combination of the two.[2,5] The accompanying sidebar, "A Subtle XSS Bug," shows that this can be quite tricky to get right.

COMPLEX, DIFFICULT-TO-REASON-ABOUT DATA FLOWS

Recall that XSS arises from flows of untrustworthy, unvalidated/unescaped data into injection-prone sinks. To assert the absence of XSS bugs in an application, a security reviewer must first find all such data sinks, and then inspect the surrounding code for context-appropriate validation and escaping of data transferred to the sink. When encountering an assignment that lacks validation and escaping, the reviewer must backwards-trace this data flow until one of the following situations can be determined:

- The value is entirely under application control and hence cannot result in attacker-controlled injection.
- The value is validated, escaped, or otherwise safely constructed somewhere along the way.
- The value is in fact not correctly validated and escaped, and an XSS vulnerability is likely present.

Let's inspect the data flow into the `innerHTML` sink in code snippet (1) in figure 1a. For illustration purposes, code snippets and data flows that require investigation are shown in red. Since no escaping is applied to `sharedAlbum.title`, we trace its origin to the `albums` entity (4) in persistent storage, via Web front-end code (2). This is, however, not the data's ultimate origin—the album name was previously entered by a different user (i.e., originated in a different time context). Since no escaping

A Subtle XSS Bug

The following code snippet intends to populate a DOM element with markup for a hyperlink (an HTML anchor element):

```
var escapedCat = goog.string.htmlEscape(category);
var jsEscapedCat = goog.string.escapeString(escapedCat);
catElem.innerHTML = '<a onclick="createCategoryList(\" +
        jsEscapedCat + '\')">' + escapedCat + '</a>';
```

The anchor element's click-event handler, which is invoked by the browser when a user clicks on this UI element, is set up to call a JavaScript function with the value of `category` as an argument. Before interpolation into the HTML markup, the value of `category` is HTML-escaped using an escaping function from the JavaScript Closure Library. Furthermore, it is JavaScript-string-literal-escaped (replacing `'` with `\'` and so forth) before interpolation into the string literal within the `onclick` handler's JavaScript expression. As intended, for a value of `Flowers & Plants` for variable `category`, the resulting HTML markup is:

```
<a onclick="createCategoryList('Flowers &amp; Plants')">
      Flowers &amp; Plants</a>
```

So where's the bug? Consider a value for `category` of:

```
');attackScript();//
```

Passing this value through `htmlEscape` results in:

```
&#39;);attackScript();//
```

because `htmlEscape` escapes the single quote into an HTML character reference. After this, JavaScript-string-literal escaping is a no-op, since the single quote at the beginning of the page is *already HTML-escaped*. As such, the resulting markup becomes:

```
<a onclick="createCategoryList('&#39;);attackScript();//')">
      &#39;);attackScript();//</a>
```

When evaluating this markup, a browser will first HTML-unescape the value of the `onclick` attribute before evaluation as a JavaScript expression. Hence, the JavaScript expression that is evaluated results in execution of the attacker's script:

```
createCategoryList(");attackScript();//')
```

Thus, the underlying bug is quite subtle: the programmer invoked the appropriate escaping functions, but in the wrong order.

was applied to this value anywhere along its flow from an ultimately untrusted source, an XSS vulnerability arises.

Similar considerations apply to the data flows in figure 1b: no validation occurs immediately prior to the assignment to `window.location.href` in (5), so back-tracing is necessary. In code snippet (6), the code exploration branches: in the true branch, the value originates in a configuration entity in the data store (3) via the Web front end (8); this value can be assumed application-controlled and trustworthy and is safe to use without further validation. It is noteworthy that the persistent storage contains both trustworthy and untrustworthy data in different entities of the same schema—no blanket assumptions can be made about the provenance of stored data.

In the else-branch, the URL originates from a parameter of the *current* URL, obtained from `window.location.href`, which is an attacker-controlled source (7). Since there is no validation, this code path results in an XSS vulnerability.

MANY OPPORTUNITIES FOR MISTAKES

Figures 1a and 1b show only two small slices of a hypothetical Web application. In reality, a large, nontrivial Web application will have hundreds if not thousands of branching and merging data flows into injection-prone sinks. Each such flow can potentially result in an XSS bug if a developer makes a mistake related to validation or escaping.

Exploring all these data flows and asserting absence of XSS is a monumental task for a security reviewer, especially considering an ever-changing code base of a project under active development. Automated tools that employ heuristics to statically analyze data flows in a code base can help. In our experience at Google, however, they do not substantially increase confidence in review-based assessments, since they are necessarily incomplete in their reasoning and subject to both false positives and false negatives. Furthermore, they have similar difficulties to human reviewers with reasoning about whole-system data flows across multiple system components, using a variety of programming languages, RPC (remote procedure call) mechanisms, and so forth, and involving flows traversing multiple time contexts across data stores.

Similar limitations apply to dynamic testing approaches: it is difficult to ascertain whether test suites provide adequate coverage for whole-system data flows.

TEMPLATE SYSTEMS TO THE RESCUE?

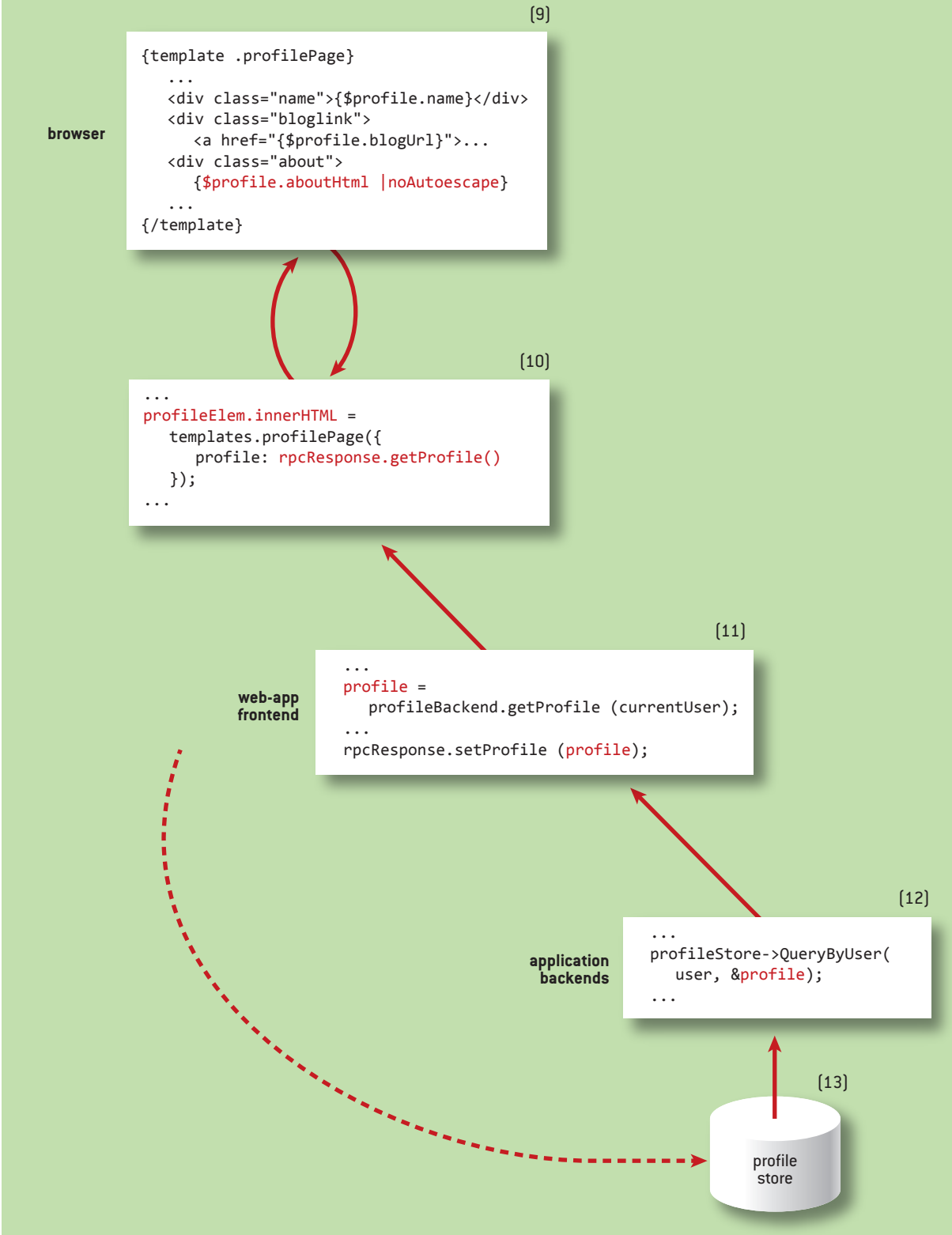
In practice, HTML markup, and interpolation points therein, are often specified using *HTML templates*. Template systems expose domain-specific languages for rendering HTML markup. An HTML markup template induces a function from template variables into strings of HTML markup.

Figure 1c illustrates the use of an HTML markup template (9): this example renders a user profile in the photo-sharing application, including the user's name, a hyperlink to a personal blog site, as well as free-form text allowing the user to express any special interests.

Some template engines support automatic escaping, where escaping operations are automatically inserted around each interpolation point into the template. Most template engines' auto-escape facilities are noncontextual and indiscriminately apply HTML escaping operations, but do not account for special HTML contexts such as URLs, CSS, and JavaScript.

Contextually auto-escaping template engines[6] infer the necessary validation and escaping operations required for the context of each template substitution, and therefore account for such special contexts.

FIGURE 1C **Using an HTML Markup Template**



Use of contextually auto-escaping template systems dramatically reduces the potential for XSS vulnerabilities: in (9), the substitution of the untrustworthy values `profile.name` and `profile.blogUrl` into the resulting markup cannot result in XSS—the template system automatically infers the required HTML-escaping and URL-validation.

XSS bugs can still arise, however, in code that does not make use of templates, as in figure 1a (1), or that involves non-HTML sinks, as in figure 1b (5).

Furthermore, developers occasionally need to exempt certain substitutions from automatic escaping: in figure 1c (9), escaping of `prof le.aboutHtml` is explicitly suppressed because that field is assumed to contain a user-supplied message with simple, safe HTML markup (to support use of fonts, colors, and hyperlinks in the "about myself" user-profile field).

Unfortunately, there is an XSS bug: the markup in `profile.aboutHtml` ultimately originates in a rich-text editor implemented in browser-side code, but there is no server-side enforcement preventing an attacker from injecting malicious markup using a tampered-with client. This bug could arise in practice from a misunderstanding between front-end and back-end developers regarding responsibilities for data validation and sanitization.

RELIABLY PREVENTING THE INTRODUCTION OF XSS BUGS

In our experience at Google's security team, code inspection and testing do not ensure, to a reasonably high degree of confidence, the absence of XSS bugs in large web applications. Of course, both inspection and testing provide tremendous value and will typically find *some* bugs in an application (perhaps even *most* of the bugs), but it is hard to be sure whether or not they discovered *all* the bugs (or even *almost all* of them).

Can we do better? We think so. Over the past few years security engineers at Google have developed practical software design patterns that make the development of Web applications much more resistant to inadvertent introduction of XSS vulnerabilities into application code, as compared with current practice.

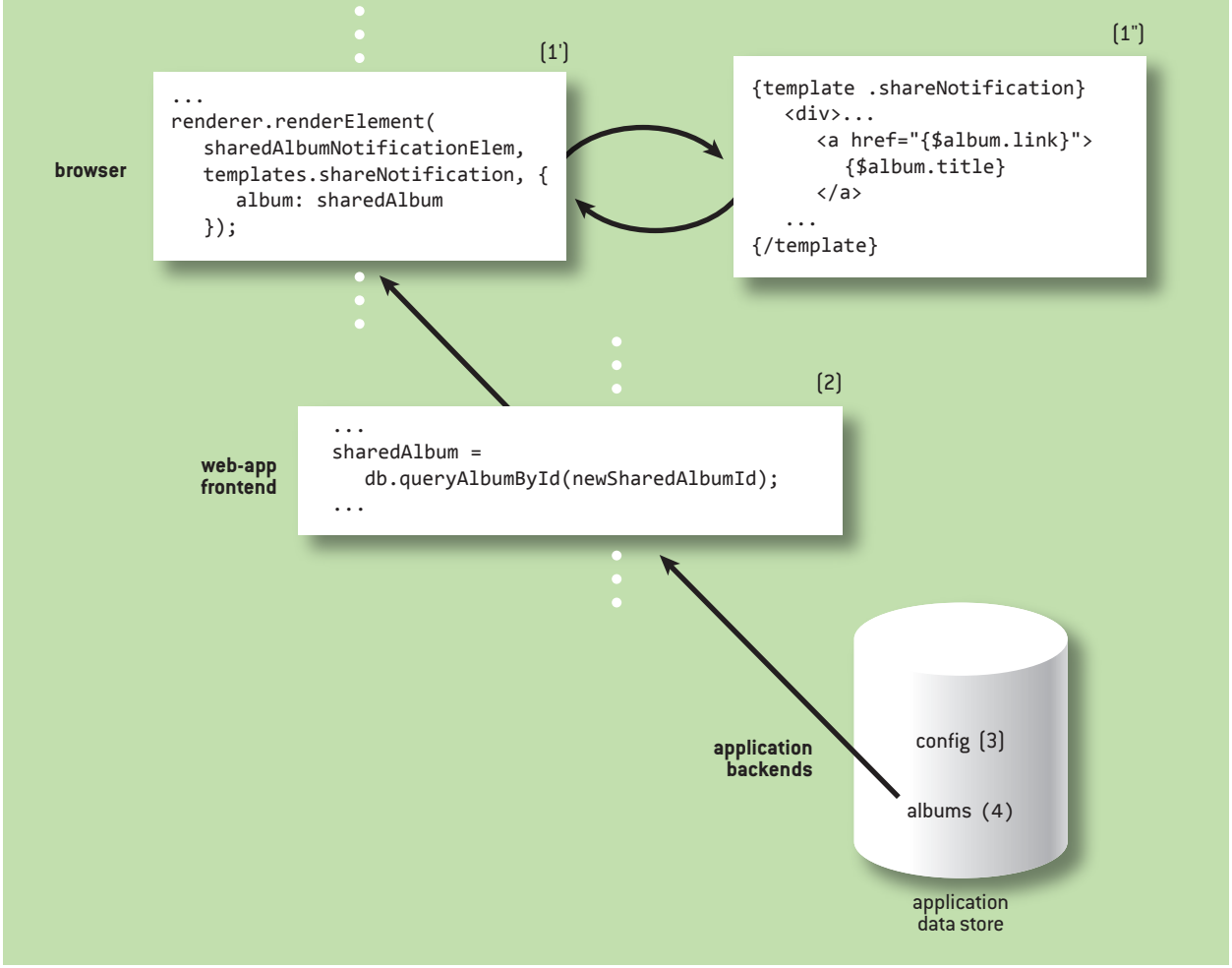
The primary goal of this approach is to limit code that could potentially give rise to XSS vulnerabilities to a very small fraction of an application's code base.

A second, equally important goal is to provide a developer experience that does not add an unacceptable degree of friction as compared with existing developer workflows.

Key components of this approach are:

- **Inherently safe APIs.** Injection-prone Web-platform and HTML-rendering APIs are encapsulated in wrapper APIs designed to be inherently safe against XSS in the sense that no use of such APIs can result in XSS vulnerabilities.
- **Security type contracts.** Special types are defined with contracts stipulating that their values are safe to use in specific contexts without further escaping and validation.
- **Coding guidelines.** Coding guidelines restrict direct use of injection-prone APIs, and ensure security review of certain security-sensitive APIs. Adherence to these guidelines can be enforced through simple static checks.

FIGURE 2A **Replacing Ad-Hoc Concatenation of HTML Markup with a Strict Template**



INHERENTLY SAFE APIS

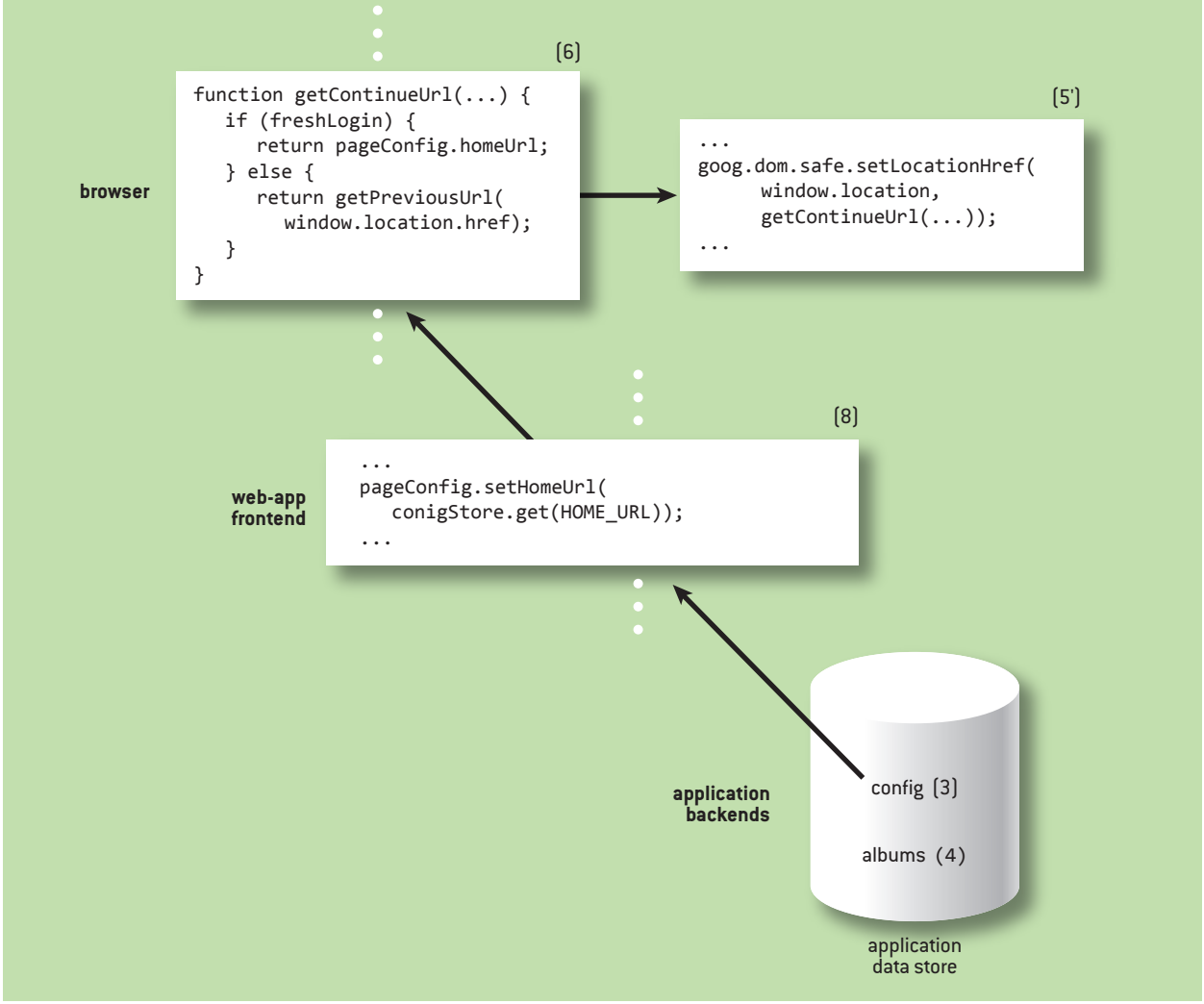
Our goal is to provide inherently safe wrapper APIs for injection-prone browser-side Web platform API sinks, as well as for server- and client-side HTML markup rendering.

For some APIs, this is straightforward. For example, the vulnerable assignment in figure 1b (5) can be replaced with the use of an inherently safe wrapper API, provided by the JavaScript Closure Library, as shown in figure 2b (5′). The wrapper API validates at runtime that the supplied URL represents either a scheme-less URL or one with a known benign scheme.

Using the safe wrapper API ensures that this code will not result in an XSS vulnerability, regardless of the provenance of the assigned URL. Crucially, none of the code in (5′) nor its fan-in in (6-8) needs to be inspected for XSS bugs. This benefit comes at the very small cost of a runtime validation that is technically unnecessary if (and only if) the first branch is taken—the URL obtained from the configuration store is validated even though it is actually a trustworthy value.

In some special scenarios, the runtime validation imposed by an inherently safe API may be too strict. Such cases are accommodated via variants of inherently safe APIs that accept types with a

FIGURE 2B **A Safe Wrapper API**



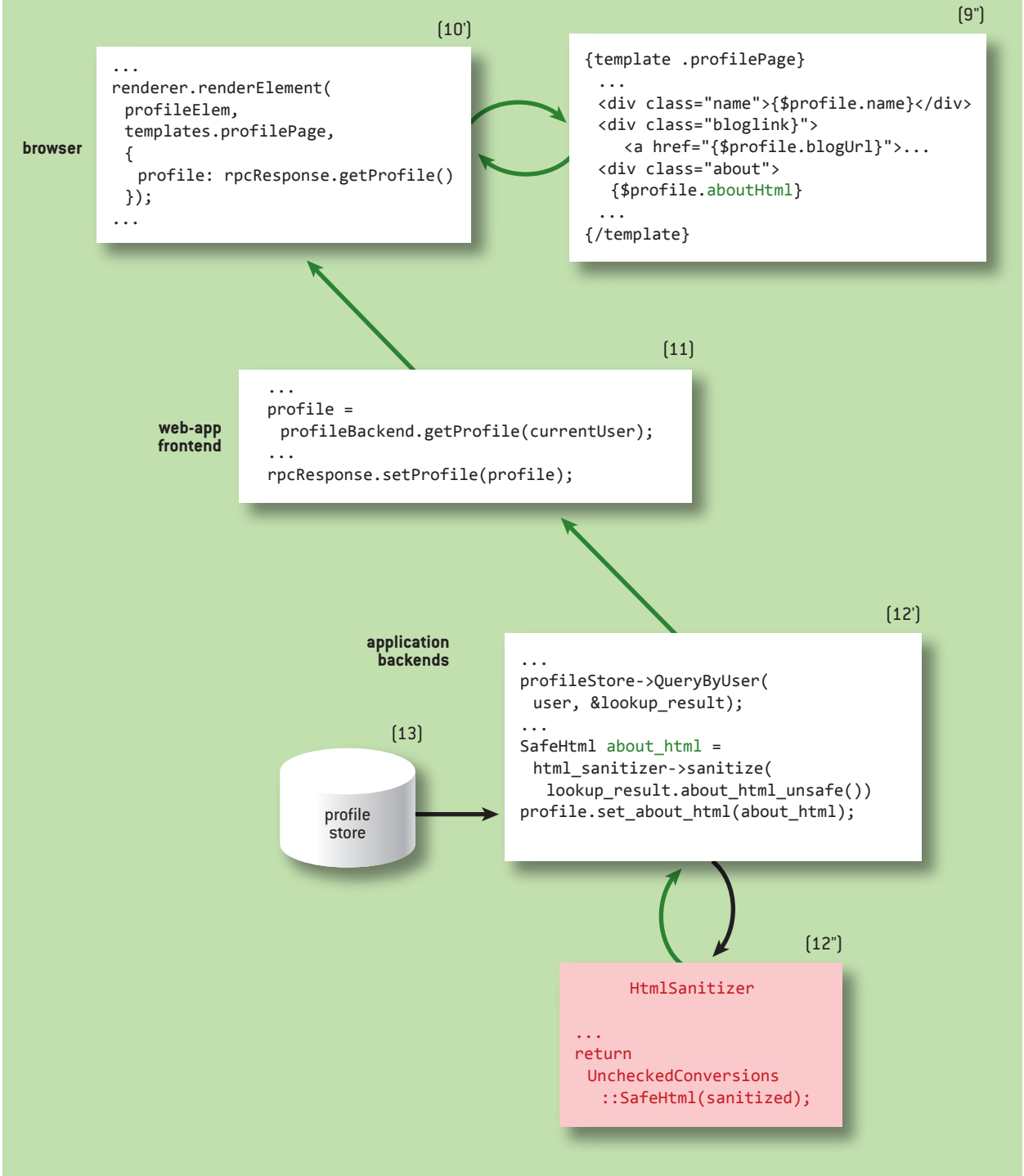
security contract appropriate for the desired use context. Based on their contracts, such values are exempt from runtime validation. This approach is discussed in more detail in the next section.

STRICTLY CONTEXTUALLY AUTO-ESCAPING TEMPLATE ENGINES

Designing an inherently safe API for HTML rendering is more challenging. The goal is to devise APIs that guarantee that at each substitution point of data into a particular context within trusted HTML markup, data is appropriately validated, sanitized, and/or escaped, unless it can be demonstrated that a specific data item is safe to use in that context based on its provenance or prior validation, sanitization, or escaping.

These inherently safe APIs are created by strengthening the concept of contextually auto-escaping template engines[6] into SCAETEs (*strictly* contextually auto-escaping template engines). Essentially, a SCAETE places two additional constraints on template code:

FIGURE 2C **Using a Type to Represent Safe HTML Markup**



• Directives that disable or modify the automatically inferred contextual escaping and validation are not permitted.

• A template may use only sub-templates that recursively adhere to the same constraint.

SECURITY TYPE CONTRACTS

In the form just described, SCAETEs don't account for scenarios where template parameters are intended to be used without validation or escaping, such as `aboutHtml` in figure 1c—the SCAETE unconditionally validates and escapes all template parameters, and disallows directives to disable the auto-escaping mechanism.

Such use cases are accommodated through types whose contracts stipulate their values are safe to use in corresponding HTML contexts, such as "inner HTML," hyperlink URLs, executable resource URLs, and so forth. Type contracts are informal: a value satisfies a given type contract if it is known that it has been validated, sanitized, escaped, or constructed in a way that guarantees that its use in the type's target context will not result in attacker-controlled script execution. Whether this is indeed the case is established by expert reasoning about code that creates values of such types, based on expert knowledge of the relevant behaviors of the Web platform.[7] As will be seen, such security-sensitive code is encapsulated in a small number of special-purpose libraries; application code uses those libraries but is not itself relied upon to correctly create instances of such types and hence does not need to be security-reviewed.

The following are examples of types and type contracts in use:

- **`SafeHtml`.** A value of type SafeHtml, converted to a string, will not result in attacker-controlled script execution when used as HTML markup.
- **`SafeUrl`.** Values of this type will not result in attacker-controlled script execution when dereferenced as hyperlink URLs.
- **`TrustedResourceUrl`.** Values of this type are safe to use as the URL of an executable or "control" resource, such as the `src` attribute of a `<script>` element, or the source of a CSS style sheet. Essentially, this type promises that its value is the URL of a resource that is itself trustworthy.

These types are implemented as simple wrapper objects containing the underlying string value. Type membership in general cannot be established by runtime predicate, and it is the responsibility of the types' security-reviewed factory methods and builders to guarantee the type contract of any instance they produce. Type membership can be based on processing (e.g., validation or sanitization), construction, and provenance, or a combination thereof.

SCAETEs use security contracts to designate exemption from automatic escaping: a substituted value is not subject to runtime escaping if the value is of a type whose contract supports its safe use in the substitution's HTML context.

Templates processed by a SCAETE give rise to functions that guarantee to emit HTML markup that will not result in XSS, assuming template parameters adhere to their security contracts, if applicable. Indeed, the result of applying a SCAETE-induced template function itself satisfies the `SafeHtml` type contract.

Figure 2c shows the application of SCAETE and security type contracts to the code slice of figure 1c. Strict contextual escaping of the template in (9) disallows use of the `noAutoescape` directive. Simply removing it, however, would enable the automatic escaping of this value, which in this case is undesired. Instead, we change the `aboutHtml` field of the `prof 1e` object to have the `SafeHtml` type, which is exempt from automatic escaping. The use of this type is threaded through the system (indicated by the color green), across RPCs all the way to the value's origin in back-end code (12').

UNCHECKED CONVERSIONS

Of course, eventually we need to create the required value of type `SafeHtml`. In the example, the corresponding field in persistent storage contains HTML markup that may be maliciously supplied by an attacker. Passing this untrusted markup through an HTML sanitizer, to remove any markup that may result in script execution, renders it safe to use in the HTML context and thus produces a value that satisfies the `SafeHtml` type contract.

To actually create values of these types, *unchecked conversion* factory methods are provided that consume an arbitrary string and return an instance of a given wrapper type (e.g., `SafeHtml` or `SafeUrl`) without applying any runtime sanitization or escaping.

Every use of such unchecked conversions must be carefully security-reviewed to ensure that in all possible program states, strings passed to the conversion satisfy the resulting type's contract, based on context-specific processing or construction. As such, unchecked conversions should be used as rarely as possible, and only in scenarios where their use is readily reasoned about for security-review purposes.

For example, in figure 2c, the unchecked conversion is encapsulated in a library (12'') along with the HTML sanitizer implementation on whose correctness its use depends, permitting security review and testing in isolation.

CODING GUIDELINES

For this approach to be effective, it must ensure that developers never write application code that directly calls potentially injection-prone sinks, and that they instead use the corresponding safe wrapper API. Furthermore, it must ensure that uses of unchecked conversions are designed with reviewability in mind, and are in fact security-reviewed. Both constraints represent coding guidelines with which all of an application's code base must comply.

In our experience, automated enforcement of coding guidelines is necessary even in moderate-size projects—otherwise, violations are bound to creep in over time.

At Google we use the open-source error-prone static checker[1] (https://code.google.com/p/error-prone/), which is integrated into Google's Java tool chain, and a feature of Google's open-source Closure Compiler (https://developers.google.com/closure/compiler/) to whitelist uses of specific methods and properties in JavaScript. Errors arising from use of a "banned" API include references to documentation for the corresponding safe API, advising developers on how to address the error. The review requirement for uses of unchecked conversions is enforced via a package-visibility mechanism provided by Google's distributed build system.[3]

If tool-chain-integrated checks were not available, coding guidelines could be enforced through simpler lint-like tools.

In the photo-sharing example, such checks would raise errors for the assignments to `innerHTML` and `location.href` in figures 1a-1c and would advise the developer to use a corresponding inherently safe API instead. For assignments to `innerHTML`, this typically means replacing ad-hoc concatenation of HTML markup with a strict template, rendered directly into the DOM element by the template system's runtime, as shown in figure 2a.

PUTTING IT ALL TOGETHER

Revisiting the code slices of the example applications after they've been brought into adherence with the coding guideline shows (figures 2a-2c) that uses of injection-prone data sinks have been replaced

with corresponding inherently safe APIs in (1′), (5′), (9′), and (10′). Now, none of these code snippets can result in an XSS bug, and neither they nor their fan-ins need to be inspected during a security review.

The only piece of code left requiring security code review (aside from infrastructure code such as the implementation of the SCAETE, its runtime, and API wrapper libraries) is the `HtmlSanitizer` package (12′′), and specifically the package-local fan-in into the unchecked conversion to `SafeHtml`. The correctness of this conversion relies solely on the correctness of the HTML sanitizer, and this package can be security-reviewed and tested in isolation. If a library is shared across multiple applications, its review cost is amortized among users.

Of course, there are limitations to the guarantees this approach can provide: first, the security reviewer may miss bugs in the security-relevant portion of the code (template systems, markup sanitizers, and so forth); second, application code may use constructs such as reflection that violate encapsulation of the types we rely on; finally, some classes of XSS bugs (in practice, relatively rare) cannot be addressed by generally applied contextual data validation and escaping as ensured by our design patterns, and these need to be addressed at other layers in Web application frameworks or in the design of individual Web applications.[7]

DEVELOPER IMPACT

Comparing the vulnerable code slices in figures 1a-1c with their safe counterparts in figures 2a-2c shows that our approach does not impose significant changes in developer workflow, nor major changes to code. For example, in figure 2b (5′), we simply use a safe wrapper instead of the "raw" Web-platform API; otherwise, this code and its fan-in remain unchanged.

The coding guidelines do require developers to use safe APIs to generate HTML markup, such as the strict template in figure 2a (1′). In return, however, using such APIs prevents XSS bugs and largely relieves developers from thinking about and explicitly specifying escaping and data validation.

Only in figure 2c is a more significant change to the application required: the type of the `aboutHtml` field changes from `String` to `SafeHtml`, and use of this type is threaded through RPCs from back end to front end. Even here, the required changes are relatively confined: a change in the field's type and the addition of a call to the `HtmlSanitizer` library in back end code (12′).

Such scenarios tend to be rare in typical Web applications. In the vast majority of uses the automatic runtime validation and escaping is functionally correct: most values of data flows into user-interface markup, both application-controlled and user-input-derived, tend to represent plain text, regular `http/https` URLs, and other values that validate and/or escape cleanly.

PRACTICAL APPLICATION

This design pattern has been applied in several open-source and proprietary Web application frameworks and template engines in use at Google: support for strict contextual auto-escaping has been added to Closure Templates (https://developers.google.com/closure/templates/), AngularJS (https://docs.angularjs.org/api/ng/service/$sce), as well as a Google-proprietary templating system. Security engineers and infrastructure developers at Google have also implemented libraries of types such as `SafeHtml`, `SafeUrl`, etc., and added inherently safe APIs to the Google Web Toolkit (http://www.gwtproject.org/doc/latest/DevGuideSecuritySafeHtml.html), the JavaScript Closure Library

(https://developers.google.com/closure/library/), and various Google-proprietary libraries and frameworks.

DECREASE IN INCIDENCE OF XSS BUGS

It is challenging to derive precise statistics regarding the impact of any particular approach to bug prevention: our design patterns prevent XSS bugs from being introduced in the first place, but we don't know how many bugs would have been introduced without their use.

We can, however, make observations based on bug counts in existing large projects that adopted our approach over time. Such observations can be considered anecdotal only, since bug counts are likely influenced by many variables such as code size and complexity, security-related developer education, etc. Nevertheless, the observations suggest that our approach significantly contributes to notable reductions in XSS vulnerabilities.

Several development teams of flagship Google applications have adopted these design patterns and coding guidelines. They have established static enforcement that all HTML markup is produced by strictly contextually auto-escaped templates, and they have disallowed direct use of certain injection-prone Web-platform APIs such as `innerHTML`.

One of the largest and most complex of these applications, using more than 1,000 HTML templates in the Closure Templates language, migrated to strict auto-escaping in early 2013. Throughout 2012 (before migration), 31 XSS bugs were filed in Google's bug tracker against this application. Post-migration, only four XSS bugs were filed in the year to mid-2014, and none at all in the first half of 2014. For another large application (also using more than 1,000 templates) whose migration is still in progress, there was a reduction from 21 to nine XSS bugs during the same time period.

Even without full compliance with the coding guidelines, some benefits can be realized: as the fraction of compliant code increases, the fraction of code that could be responsible for vulnerabilities shrinks, and confidence in the absence of bugs increases. While there is little reason not to write new code entirely in adherence to the guidelines, we can choose not to refactor certain existing code if the cost of refactoring exceeds the benefits and if we already have confidence in that code's security through other means (e.g., intensive review and testing).

CONCLUSION

Software design can be used to isolate the potential for XSS vulnerabilities into a very small portion of an application's code base. This makes it practical to intensively security-review and test just those portions of the code, resulting in a high degree of confidence that a Web application as a whole is not vulnerable to XSS bugs. Our approach is practically applicable to large, complex, real-world Web applications, and it has resulted in significant reduction of XSS bugs in several development projects.

This approach to what is fundamentally a difficult problem involving whole-system data flows incorporates two key principles:

• Based on the observation that in typical Web apps, it is functionally correct to conservatively runtime-escape and -validate the vast majority of data flowing into injection-prone sinks, we choose to treat all string-typed values as potentially untrustworthy and subject to runtime validation and escaping, regardless of their provenance. This design choice altogether obviates the need for whole-program reasoning about the vast majority of whole-system data flows in a typical Web application.

• Only in scenarios where default, runtime validation and escaping is functionally incorrect, we employ type contracts to convey that certain values are already safe to use in a given context. This use of types permits compositional reasoning about whole-system data flows and allows security experts to review security-critical code in isolation, based on package-local reasoning.

Our coding guidelines impose certain constraints on application code (though they typically require only limited changes to existing code). In contrast, many existing approaches to the prevention and detection of XSS aim to be applicable to existing, unmodified code. This requirement makes the problem much more difficult, and generally requires the use of complex whole-program static and/or dynamic data-flow analysis techniques. For an overview of existing work in this area, see Mike Samuel et al.[6] Relaxing this requirement removes the need for special-purpose tools and technologies (such as runtime taint tracking or whole-program static analysis), allowing us to rely solely on the combination of software design, coding guidelines enforceable by very simple static checks, existing language-native type systems, and a small enhancement to existing contextually auto-escaping template systems. Thus, our approach can be used in applications written in a variety of programming languages, without placing special requirements on tool chains, build systems, or runtime environments.

REFERENCES

1. Aftandilian, E., Sauciuc, R., Priya, S., Krishnan, S. 2012. Building useful program analysis tools using an extensible Java compiler. International Working Conference on Source Code Analysis and Manipulation (SCAM): 14-23.
2. Daswani, N., Kern, C., Kesavan, A. 2007. *Foundations of Security: What Every Programmer Needs to Know*. Apress.
3. Morgenthaler, J. D., Gridnev, M., Sauciuc, R., Bhansali, S. 2012. Searching for build debt: experiences managing technical debt at Google. Third International Workshop on Managing Technical Debt (MTD): 1-6.
4. OWASP. 2013. Top 10 List; https://www.owasp.org/index.php/Top_10_2013-Top_10.
5. OWASP. 2014. XSS (cross site scripting) prevention cheat sheet; https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet.
6. Samuel, M., Saxena, P., Song, D. 2011. Context-sensitive auto-sanitization in Web templating languages using type qualifiers. *Proceedings of the 18th ACM Conference on Computer and Communications Security:* 587-600.
7. Su, Z. Wassermann, G. 2006. The essence of command injection attacks in web applications. POPL '06. http://dl.acm.org/citation.cfm?id=1111070
8. Zalewski, M. 2011. *The Tangled Web: A Guide to Securing Modern Web Applications*. No Starch Press.

LOVE IT, HATE IT? LET US KNOW

feedback@queue.acm.org

CHRISTOPH KERN (xtof@google.com) is an information security engineer at Google. His primary focus is on designing APIs and frameworks that make it easy for developers to write secure software and eliminate or reduce the risk of developers accidentally introducing security bugs.