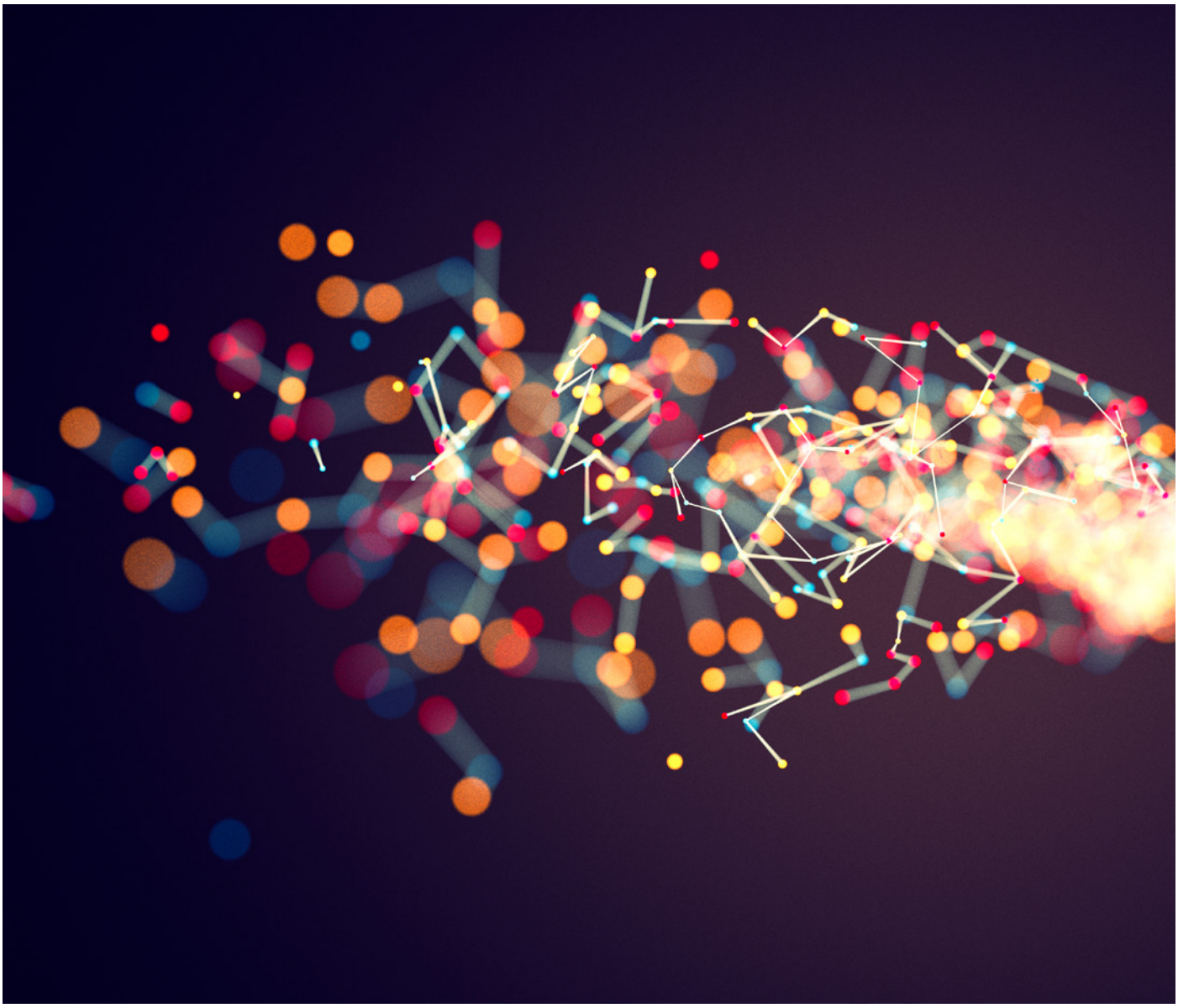
THE MORNING PAPER
QUARTERLY REVIEW

REDUNDANCY DOES NOT IMPLY FAULT TOLERANCE

WHEN DNNS GO WRONG

THE CURIOUS CASE OF THE PDF CONVERTER THAT LIKES MOZART



A selection of papers from the world of computer science, as featured by Adrian Colyer on The Morning Paper blog

InfoQ queue

CONTENTS

Issue # 5, June 2017

WELCOME...

Welcome to the latest edition of The Morning Paper quarterly review. I've chosen five posts for you that appeared on the blog (https://blog.acolyer.org) in the first quarter of 2017. This time around, they actually represent more than five papers though, as one of these posts is itself a collection of shorter summaries of a number of papers on a theme. My five selections are:

Adrian Colyer

TOWARD SUSTAINABLE INSIGHTS, OR WHY POLYGAMY IS BAD FOR YOU

Binning et al., CIDR 2017

If you've ever wondered about all those headlines touting 'New study reveals X causes Y' then this is the paper for you. It's a look at something called the *multiple comparisons problem* and how data exploration tools can inadvertently lead you to think that you've found a true correlation in your data, when in fact you're just looking at random chance. QUDE is a data exploration tool with built in support to help you avoid such common statistical mistakes. Don't worry, I take things really gently in this write-up - it's the only way I can avoid confusing myself! In the piece itself, I talk about 0.05 as the p-value for statistical significance. This is common in social sciences, but different domains do use different values. Derek Jones has an interesting discussion of p-values in software engineering on his blog.

WHEN DNNS GO WRONG - ADVERSARIAL EXAMPLES AND WHAT WE CAN LEARN FROM THEM

This piece is actually a summary of seven different papers on the theme of *adversarial examples*. An adversarial example is a deliberately constructed input that can cause a deep neural network to produce a wrong (unexpected output). For example, we can make an image classifier look at a picture of a panda and predict with high confidence it is a gibbon. Take those attacks out into the real world - for example, carefully defacing road signs - and things start to get really interesting. Understanding adversarial examples teaches us something about the differences between how DNNs see the world, and how we see it. As I write this, there are no known defences against adversarial examples.

Subsequent to my write-up, a team from the Bosch Center for Artificial Intelligence, in conjunction with the University of Freiburg published a paper showing that can create *universal* perturbations that

can make a network yield a desired outcome[1]. I think of it a little bit like optical illusions are to humans. We'll be covering this work in a future edition of The Morning Paper.

[1] Universal adversarial perturbations against semantic image segmentation.

THOU SHALT NOT DEPEND ON ME: ANALYSING THE USE OF OUTDATED JAVASCRIPT LIBRARIES ON THE WEB

Lauinger et al., NDSS 2017

It's the simplest thing in the world to add a new client-side JavaScript library to your website. Keeping track of all those dependencies and ensuring they stay up-to-date and vulnerability free is another issue altogether. Moreover, many libraries you include in your site in turn pull in other dependencies, and there may be vulnerabilities in those too! The authors of this paper conduct an extensive survey and find out that over a third of websites include at least one library with a known vulnerability. Furthermore, included libraries can often be years behind the most recent releases. It seems 'include and forget' is a pretty good description of current practice in many organisations. We can, and should, do better.

REDUNDANCY DOES NOT IMPLY FAULT TOLERANCE: ANALYSIS OF DISTRIBUTED STORAGE REACTIONS TO SINGLE ERRORS AND CORRUPTIONS

Ganesan et al., FAST 2017

Network partitions and file system corruptions. Two things that developers of large-scale datastores would like to pretend don't happen very often. Thanks to the work of Kyle Kingsbury and Jepsen.io, awareness of the implications of network partitions and the level of rigour needed to ensure consistency (according the to level of promises made by the datastore in question) has risen greatly. Awareness of the issues relating to file system errors and corruptions is much lower though. This paper changes that, showing us that even a single file system fault can induce catastrophic outcomes in most modern distributed storage systems.

THE CURIOUS CASE OF THE PDF CONVERTER THAT LIKES MOZART: DISSECTING AND MITIGATING THE PRIVACY RISK OF PERSONAL CLOUD APPS

Harkous et al., PoPET'16

We've all seen important privacy information buried deep in lengthy terms and conditions documents that no-one ever reads. New legislation (for example, the GDPR in Europe) is seeking to put an end to this practice, and instead requires companies to gather explicit consent in as clear and simple a manner as possible. In this paper, Harkous et al. investigate the design of privacy notification dialogs designed to help people understand what they're really consenting to. You won't be surprised to learn that many people just click-through standard permissions dialogs without paying too much attention, even when they ask for way more access than the app actually needs. The 'Far Reaching Insights' approach to privacy notices really does get people's attention though. I think it's brilliant, and I look forward to seeing a lot more of this sort of thing in the wild.

TOWARD SUSTAINABLE INSIGHTS, OR WHY POLYGAMY IS BAD FOR YOU

Buckle up! Today we're going to be talking about statistics, p-values, and the multiple comparisons problem.

Binning et al., CIDR 2017

Some good background resources here are:

- Statistics Done Wrong, by Alex Reinhart
- p-values on wikipedia
- Misunderstandings of p-values, also on wikipedia

For my own benefit, I'll try and explain what follows as simply as possible – I find it incredibly easy to make mistakes otherwise! Let's start with a very quick recap of p-values.

P-VALUES

If we observe some variable X and see value x, we might wonder "what are the odds of that!" If we knew the underlying probability distribution for X we'd be able to give an answer. For example, you roll a fair dice and get a 1. We know that the odds of that are 1 in 6 (1/6). If we don't know the underlying probability distribution we can't the answer the question at all. But we *can* answer a related question: '*suppose* we're seeing the results of a fair dice roll, what would be the odds of getting a 1?' And the answer is of course 1/6 again. Let's call 'suppose we're seeing the results of a fair dice roll' our *hypothesis* H about the underlying distribution. Then what we're really asking is the probability that an observation of X will be x given that hypothesis, or : $P(X = x|H)$. We call the probability of a given observation given an underlying hypothesis a *p-value*.

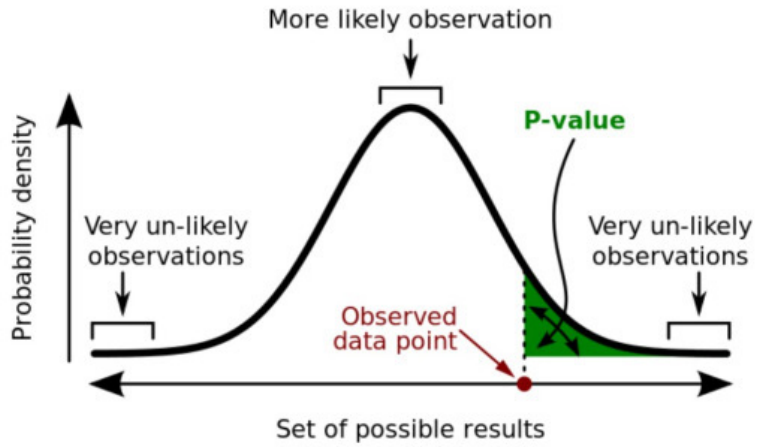
Time to move on from dice rolls. Suppose the variable X we observe is now a measure of correlation between two measured phenomena. That's a continuous variable, so instead of asking is X exactly equal to some value x we need to ask 'what are the odds of seeing a value $\leq x$ (or

$\geq x$)?' We know that we can't answer that question unless it's in relation to some underlying probability distribution. We need a hypothesis H. For correlation questions we often use the *null hypothesis*, which is the hypothesis that the two measured phenomena are in fact completely independent.

> ***Suppose we see a suspiciously large value. What are the odds of that?***

Suppose we see a suspiciously large value. What are the odds of that?

p-value = $P(X \geq x|H)$



A **p-value** (shaded green area) is the probability of an observed (or more extreme) result assuming that the null hypothesis is true.

Example of a *p*-value computation. The vertical coordinate is the probability density of each outcome, computed under the null hypothesis. The *p*-value is the area under the curve past the observed data point.

(source: wikipedia)

Here's the first thinking trap. The p-value tells us the likelihood of seeing a given value (greater than or equal to in this example) of X given some hypothesis. We cannot swap the order and treat it as the probability that the hypothesis is true given our observation!

$$P(observation|hypothesis) \neq P(hypothesis|observation)$$

An arbitrary but universally accepted p-value of 0.05 (there's a 5% chance of this observation given the hypothesis) is deemed as the threshold for 'statistical significance.' Really, things don't divide neatly into 'statistically significant' and 'statistically insignificant', there's just a continuous underlying probability. p=0.051 is not dramatically different from p=0.049 even though one is 'significant' and the other isn't.

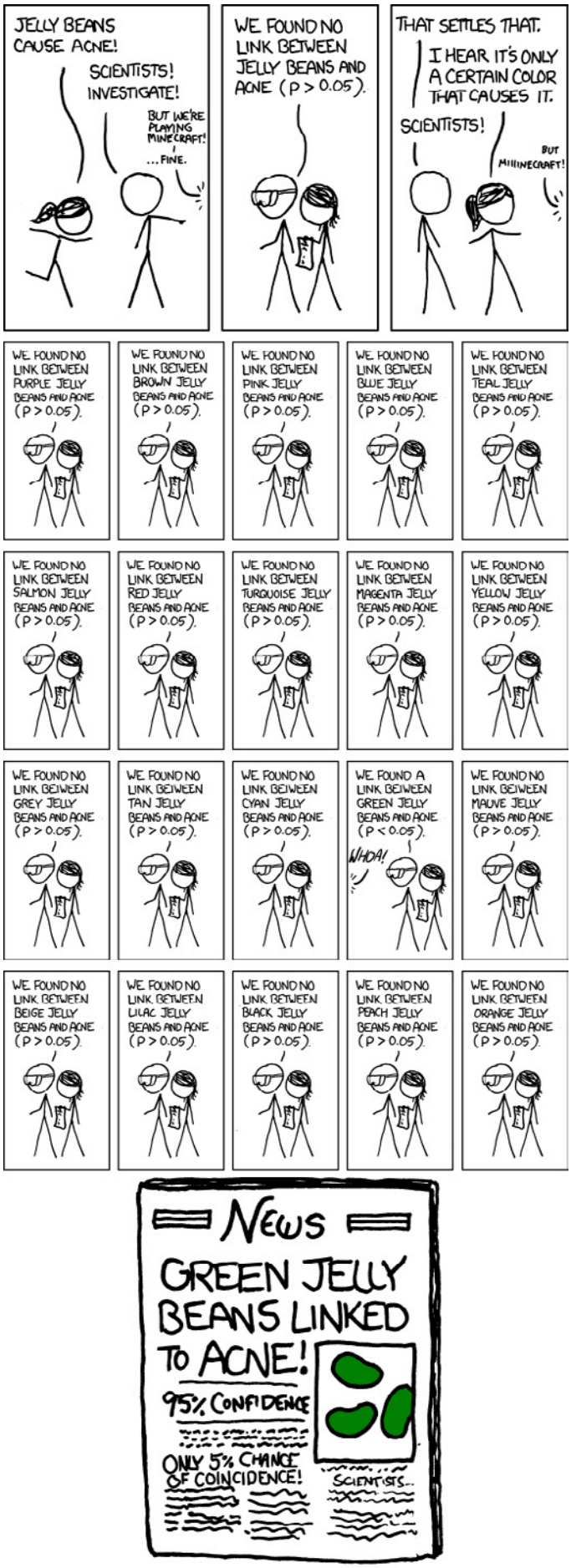
MULTIPLE COMPARISONS

Take a look at this xkcd cartoon. When we look at the correlation between green jelly beans and acne, we find a statistically significant result (i.e., $p < 0.05$). Publish!!

But hold on, there's some kind of weird 'Schrödinger statistics' that we have to take into account! The true significance of our green jelly bean finding depends on how many other jelly bean colours we also examined for correlation with acne before we looked at the green beans. Prior observations change the world. Sounds like nonsense doesn't it! How can the fact that I previously looked into blue beans in any way change the significance of the green bean correlation with acne – the numbers are there in black and white and don't change whatever I might have looked at before?!

We've been tricked into thinking we've answered one question 'what's the significance of the correlation between green jelly beans and acne?', when what we're really doing is answering the following question: 'what's the chance of finding at least one jelly bean colour that shows a statistically significant correlation with acne, even though the two phenomena are actually independent?' It matters a lot whether 'green' is something we discovered (we're answering the latter question), or something we fixed a priori (the former question).

It helps me a lot to think of it this way: imagine a population of 1000 points, in a normal distribution. If you sample a point at random, there's a slim chance that you picked an outlier point. Not knowing the underlying distribution, you'd say that there was a low chance (i.e., statistically significant) that this really came from the true underlying distribution (even though it did of course). But what if you keep on exploring the space, repeatedly sampling points until you've taken e.g., 10,000 samples. What are the chances you'll catch some of those



outlier points in your sample? Incredibly high! Now you could make the true statement "I took a sample at random and found a point which is unlikely to have come from such an underlying distribution with statistical significance." Such a statement can be very misleading though unless you also qualify it with how many points you looked at before you found it.

Let's take a concrete example. Suppose under the null hypotheses there's a 99% chance you'll observe a value of $X \leq x$. You observe X and lo and behold see a value $\leq x$, this is a statistically significant result with p-value 0.01 (1% chance). After n trials, the odds of *not* seeing such an outlier are simply 0.99^n. After 6 trials, that's 0.94, or a 94% chance of not seeing an outlier. That means of course, that after 6 trials there's a 6% chance you *will* see an outlier. That's a p-value of 0.06, so with just 6 trials (or any higher number of trials), it is *statistically insignificant* that you saw an outlier point!

MULTIPLE COMPARISONS AND DATA VISUALISATION

That was a long introduction! Finally we get to the paper. We want to help scientists find interesting correlations in their data – a number of tools have grown up to help do this. The core idea is to have the tool investigate lots of potential correlations, and when one is found that seems to be 'interesting,' to show it to the user. Very helpful you say – it would take me ages to discover these things myself! But as we now know, we have to very careful we don't fall into the multiple comparisons trap. How many combinations were tried before hitting on this one? If we don't know, then we actually don't know what the statistical significance of the visualisation we're looking at is, even though it may appear in isolation to be highly significant.

... without knowing how exactly the system tried to find 'interesting' correlations and how many correlations it tested, it is later on impossible for the user to determine what the expected false discovery rate will be across the whole data exploration session.

When using an interactive visualisation tool, not only is the statistical significance of the visualised results unclear, but the more you explore the greater the odds of finding *false discoveries*.

With every additional hypothesis test the chance of finding a false discovery increases. This problem is known as the 'multiple comparisons problem' and has been studied extensively in the statistics literature.

There are some great examples of such spurious correlations found by existing tools in the opening sections of the paper.

The authors set out to build a data exploration and visualization tool called QUDE (Quantifying the Uncertainty in Data Exploration) – pronounced 'cute', which only makes sense to American readers!! QUDE integrates a 'risk detection engine' alongside the explorer to always show the user the risk that something is a false discovery.

> *Two fundamental challenges arise when attempting to automatically quantify the risk: (1) the traditional techniques either do not scale well with the number of hypothesis or can not be used in an interactive environment and (2) in many cases it is not clear which hypothesis is currently being tested through a visualization by the user (i.e., the 'user intent').*

QUDE uses a measure called the *False Discovery Rate* (FDR). If V is the number of 'Type 1' (false positive) errors in individual tests, and R is the total number of null hypotheses rejected by a multiple test, then FDR is defined as the expected ratio of erroneous rejections among all rejections $E[V/R]$. See §3.1 in the paper for details.

Beyond controlling the multiple hypothesis error, the QUDE team also plan to include support for detecting other common statistical mistakes in time, including Simpson's paradox, the Base rate fallacy, imbalance of labels and pseudoreplication.

Given the long introduction I had space only for the briefest description of QUDE itself, so please do go on to check out the full paper if this catches your interest.

I'll leave you with this opening quote from the paper:

> *A new study shows that drinking a glass of wine is just as good as spending an hour at the gym" [Fox News, 02/15]. "A new study shows how sugar might fuel the growth of cancer" [Today, 01/16]. "A new study shows late night snacking could damage the part of your brain that creates and stores memories" [Fox News, 05/16].*

We've all seen endless such stories, often contradicting each other. If we don't know how many other 'studies' researchers did before hitting on these particular results, we really don't know their significance at all. (I have no knowledge of the situation for the particular cases in the quote above).

WHEN DNNS GO WRONG

ADVERSARIAL EXAMPLES AND WHAT WE CAN LEARN FROM THEM

One way of understanding what's going on inside a network is to understand what can break it. Adversarial examples are deliberately constructed inputs which cause a network to produce the wrong outputs (e.g., misclassify an input image).

Deep neural networks are easily fooled, *Nguyen et al., 2015*
Practical black-box attacks against deep learning systems using adversarial examples, *Papernot et al., 2016*
Adversarial examples in the physical world, *Goodfellow et al., 2017*
Explaining and harnessing adversarial examples, *Goodfellow et al., 2015*
Distillation as a defense to adversarial perturbations against deep neural networks, *Papernot et al., 2016*
Vulnerability of deep reinforcement learning to policy induction attacks, *Behzadan & Munir, 2017*
Adversarial attacks on neural network policies, *Huang et al. 2017*

We'll start by looking at 'Deep Neural Networks are Easily Fooled' from the 'top 100 awesome deep learning papers list,' and then move on to some other examples cited in the excellent recent OpenAI post on "Attacking machine learning with adversarial examples."

Might I suggest *two* cups of coffee for this one...

DEEP NEURAL NETWORKS ARE EASILY FOOLED

What's this?

Clearly it's an armadillo! I'll make it easier for you... these are five different images of a digit between 0 and 9, but which one?

(It's a 4, obviously).

What you're seeing here are *adversarial* images, deliberately crafted to classify as some class *x*, while clearly looking nothing like the target class from a human perspective.

> *... it is easy to produce images that are completely unrecognizable to humans, but that state-of-the-art DNNs believe to be recognizable objects with 99.99% confidence (e.g., labelling with certainty that white noise static is a lion).*

The fact that we can do this tells us something about interesting about the differences between DNN vision and human vision. Clearly, the DNNs are not learning to interpret images in the same way that we do.

The adversarial images are created using an evolutionary algorithm (EA) that evolves a population of images. Standard EAs use a single fitness function, but the authors here use a new algorithm called MAP-Elites that allows simultaneous evolution of a population containing individuals scoring well on many classes – in each round the best individual so far for each objective is kept. Two different mutation strategies are tested: one that directly encodes pixels in grayscale and then mutates their values, and one that uses an *indirect encoding* based on a compositional pattern-producing network (CPNN) which can evolve complex regular images that resemble natural and man-made objects.

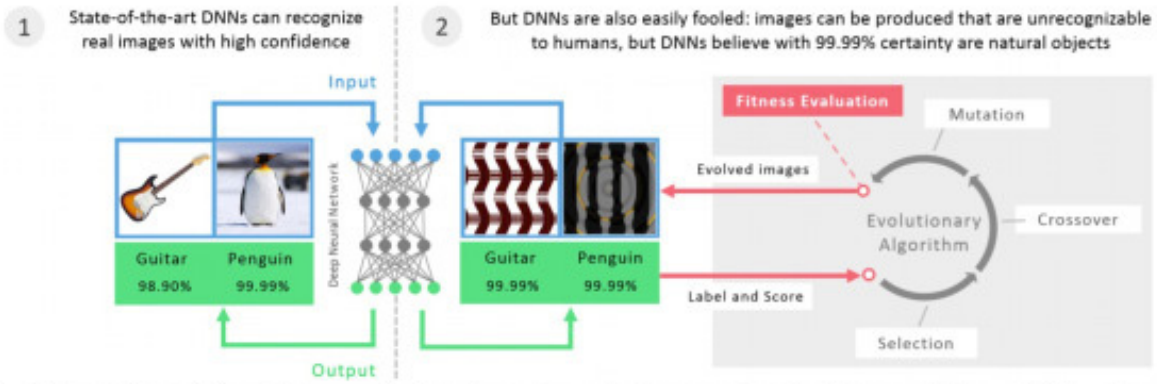


Figure 2. Although state-of-the-art deep neural networks can increasingly recognize natural images (*left panel*), they also are easily fooled into declaring with near-certainty that unrecognizable images are familiar objects (*center*). Images that fool DNNs are produced by evolutionary algorithms (*right panel*) that optimize images to generate high-confidence DNN predictions for each class in the dataset the DNN is trained on (here, ImageNet).

Take MNIST as an example (digits 0-9). Starting with clean images, within 50 generations images are produces that MNIST DNNs will misclassify with 99.99% confidence but are unrecognisable as such. These images were created used the direct encoding mutation:

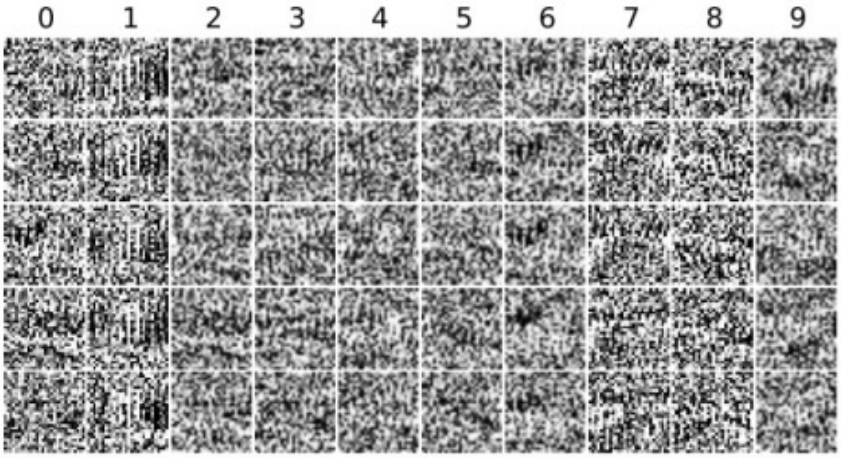


Figure 4. Directly encoded, thus irregular, images that MNIST DNNs believe with 99.99% confidence are digits 0-9. Each column is a digit class, and each row is the result after 200 generations of a randomly selected, independent run of evolution.

And these were created using the indirect encoding mutation:

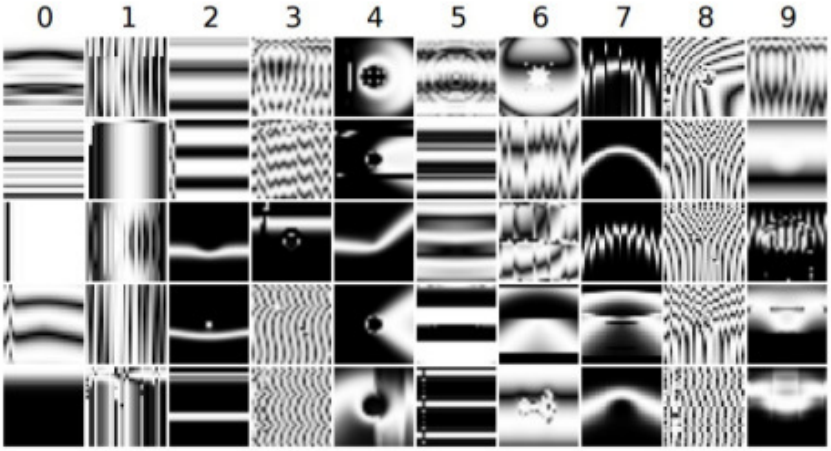


Figure 5. Indirectly encoded, thus regular, images that MNIST DNNs believe with 99.99% confidence are digits 0-9. The column and row descriptions are the same as for Fig. 4.

Using the CPNN encoding and deliberately evolving images to match target DNN classes results in a wide variety of images:

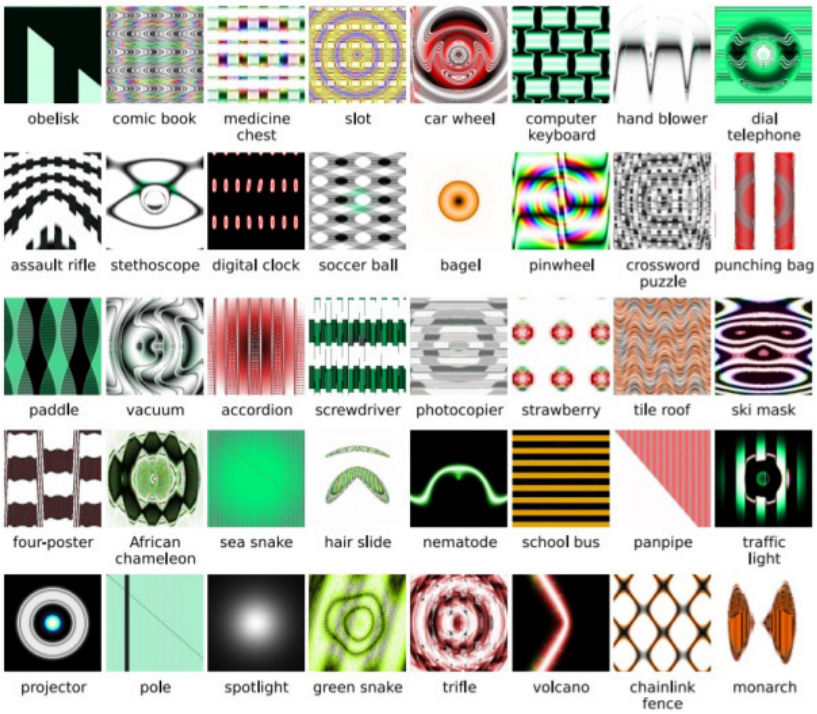


Figure 8. Evolving images to match DNN classes produces a tremendous diversity of images. Shown are images selected to showcase diversity from 5 evolutionary runs. The diversity suggests that the images are non-random, but that instead evolutions producing discriminative features of each target class. The mean DNN confidence scores for these images is 99.12%.

For many of the produced images, one can begin to identify why the DNN believes the image is of that class once given the class label. This is because evolution need only produce features that are unique to, or discriminative for, a class, rather than produce an image that contains all of the typical features of a class.

By removing some of the repeated elements from the generated images, the confidence score of the DNN drops. "*These results suggest that DNNs tend to learn low and middle-level features rather than the global structure of objects.*"

You might wonder if we can make a DNN more robust to such adversarial images by extending the training regime to include such negative examples. The authors tried this, but found that it was always possible to generate new adversarial examples that still fooled the resulting network (this remained true even after 15 iterations of the process).

Why is it so easy to generate adversarial examples? Discriminative models create decision boundaries that partition data into classification regions. In a *high-dimensional* input space, the area a model allocates to a class may be much larger than the area occupied by training examples for the class. This leaves plenty of room for adversarial images...

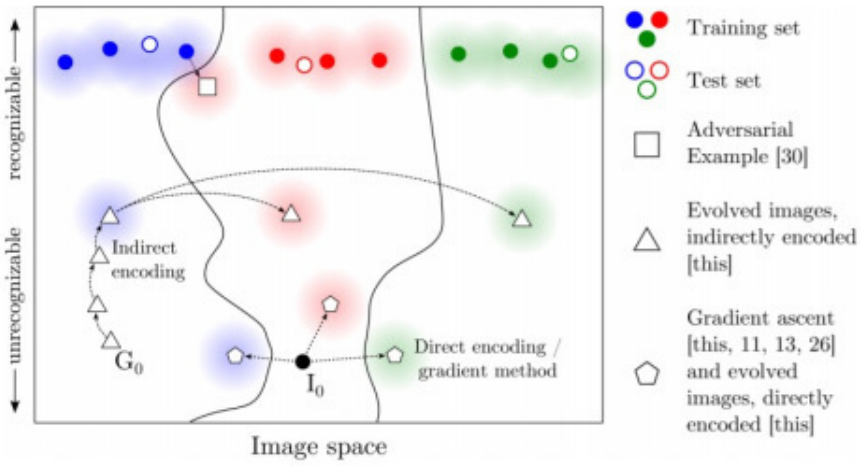


Figure 14. Interpreting our results and related research. (1) [30] found that an imperceptible change to a correctly classified natural image (blue dot) can result in an image (square) that a DNN classifies as an entirely different class (crossing the decision boundary). The difference between the original image and the modified one is imperceptible to human eyes. (2) It is possible to find high-confidence images (pentagon) using our directly encoded EA or gradient ascent optimization starting from a random or blank image (I_0) [11, 13, 26]. These images have blurry, discriminative features of the represented classes, but do not look like images in the training set. (3) We found that indirectly encoded EAs can find high-confidence, regular images (triangles) that have discriminative features for a class, but are still far from the training set.

We now turn our attention from adversarial examples as a way of *understanding* what DNNs are doing, to adversarial examples as a way of *attacking* DNNs..

The fact that DNNs are increasingly used in a wide variety of industries, including safety-critical ones such as driverless cars, raises the possibility of costly exploits via techniques that generate fooling images...

PRACTICAL BLACK-BOX ATTACKS AGAINST DEEP LEARNING SYSTEMS USING ADVERSARIAL EXAMPLES

This is a panda (59.7% confidence):

But *this* is obviously a gibbon (99.3% confidence):

(From 'Explaining and harnessing adversarial examples,' which we'll get to shortly).

The goal of an attacker is to find a small, often imperceptible perturbation to an existing image to force a learned classifier to misclassify it, while the same image is still correctly classified by a human. Previous techniques for generating adversarial images relied on either access to the full training set, and/or the hidden weights in the network. What this paper shows is that successful attacks can be mounted even without such information – all you need is the ability to pass an input to the classifier, and learn the resulting predicted class.

Our threat model thus corresponds to the real-world scenario of users interacting with classifiers hosted remotely by a third-party keeping the model internals secret. In fact, we instantiate our attack against classifiers served by MetaMind, Amazon, and Google. Models are automatically trained by the hosting platform. We are capable of making labeling prediction queries only after training is completed. Thus, we provide the first correctly blinded experiments concerning adversarial examples as a security risk.

The attack works by training a substitute model (owned by the attacker) using the target DNN as an oracle. Target inputs are synthetically generated, passed to the oracle (system under attack), and the output labels becomes the training labels for the substitute model. One the substitute DNN has been trained, adversarial images can be created that succeed against the substitute DNN, using normal white box techniques.

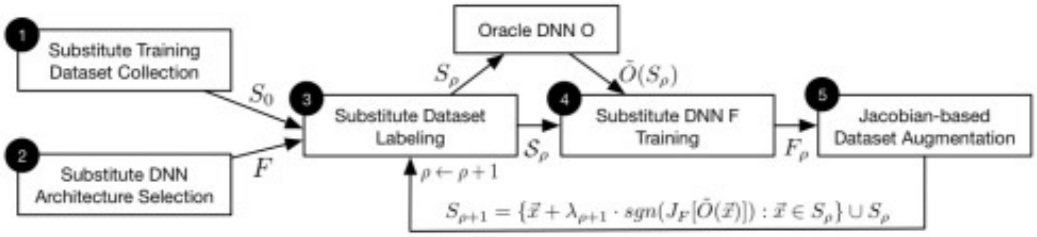


Figure 3: **Training of the substitute DNN** F: the attacker (1) collects an initial substitute training set S_0 and (2) selects an architecture F. Using oracle $\tilde{O}$, the attacker (3) labels S_0 and (4) trains substitute F. After (5) Jacobian-based dataset augmentation, steps (3) through (5) are repeated for several substitute epochs ρ.

Crucially, *the images that fool the substitute network also turn out to often force the same misclassifications in the target model.* Since the attacker only needs to (presumably) find one such image that transfers successfully this should be possible with high likelihood. It doesn't even matter if the substitute DNN has a different architecture to the target model (which it likely will, because we assume the attacker does not know the target architecture) – so long as the substitute DNN is appropriate to the kind of classification task (e.g. CNN for image classification) the attack works well. In fact, the attack doesn't only work with DNN targets – it generalizes to additional machine learning models (tested with logistic regression, SVMs, decision trees, and nearest neighbours).

The authors showed the ability to attack networks blind by using three cloud ML services provide by MetaMind, Google, and Amazon respectively. In each case training data is uploaded to the service, which learns a classifier (the user has no idea what model the service uses for this). Then the substitute network technique is used to find examples that fool the learned classifier.

An adversary using our attack model can reliably force the DNN trained using MetaMind on MNIST to misclassify 82.84% of adversarial examples crafted with a perturbation not affecting human recognition.

An Amazon classifier that achieved 92.17% test accuracy on MNIST could be fooled by 96.19% of adversarial examples. The Google classifier achieved 92% test accuracy on MNIST and could be fooled by 88.94% of adversarial examples. Defences based on gradient masking are *not effective* against the substitute attack.

ADVERSARIAL EXAMPLES IN THE PHYSICAL WORLD

So now we know that you don't need access to a model in order to successfully attack it. But there's more...

Up to now, all previous work has assumed a threat model in which the adversary can feed data directly into the machine learning classifier. This is not always the case for systems operating in the physical world, for example those which are using signals from cameras and other sensors as input. This paper shows that even in such physical world scenarios, machine learning systems are vulnerable to adversarial examples. We demonstrate this by feeding adversarial images obtained from a cell-phone camera to an ImageNet Inception classifier and measuring the classification accuracy of the system. We find that a large fraction of adversarial examples are classified incorrectly even when perceived through the camera.

The authors print clean and adversarial images, take photos of the printed images, crop those photos to be the same size as the originals, and then pass these into the classifier. The procedure takes place with manual photography and no careful control of lighting, camera angle etc., thus in introduces nuisance variability with the potential to destroy adversarial perturbations depending on subtle changes.

(a) Printout (b) Photo of printout (c) Cropped image

Figure 3: Experimental setup: (a) generated printout which contains pairs of clean and adversarial images, as well as QR codes to help automatic cropping; (b) photo of the printout made by a cellphone camera; (c) automatically cropped image from the photo.

Overall, the results show that some fraction of adversarial examples stays misclassified even after a non-trivial transformation: the photo transformation. This demonstrates the possibility of physical adversarial examples. For example, an adversary using the fast method with ε = 16 could expect that about 2/3 of the images would be top-1 misclassified and about 1/3 of the images would be top-5 misclassified. Thus by generating enough adversarial images, the adversary could expect to cause far more misclassification than would occur on natural inputs.

Other physical attacks mentioned in prior work include generation of audio inputs that mobile phones recognise as intelligible voice commands but humans hear as an unintelligible voice, and face recognition systems fooled by previously captured images of an authorized user's face...

An adversarial example for the face recognition domain might consist of very subtle markings applied to a person's face, so that a human observer would recognize their identity correctly, but a machine learning system would recognize them as being a different person.

EXPLAINING AND HARNESSING ADVERSARIAL EXAMPLES

Why do these adversarial examples work? Goodfellow et al. show us that all we need in order to be vulnerable is *linear behavior in a high-dimensional space*.

[The] results suggest that classifiers based on modern machine learning techniques, even those that obtain excellent performance on the test set, are not learning the true underlying concepts that determine the correct output label. Instead, these algorithms have built a Potemkin village that works well on naturally occuring data, but is exposed as a fake when one visits points in space that do not have high probability in the data distribution.

Consider a high-dimensional linear classifier, where the weight vector w has n dimensions. Each individual input feature has limited precision (e.g., using 8 bits per pixel in digital images, thus discarding all information below 1/255 of the dynamic range). For any one input, making a small change (smaller than the precision of the features) would not be expected to change the overall prediction of the classifier. However...

... we can make many infinitesimal changes to the input that add up to one large change to the output. We can think of this as a sort of 'accidental steganography,' where a linear model is forced to attend exclusively to the signal that aligns most closely with its weights, even if multiple signals are present and other signals have much greater amplitude.

We can maximise the impact of the many small changes by aligning the changes with the sign of the corresponding weight. This turns out to be a fast way of generating adversarial images.

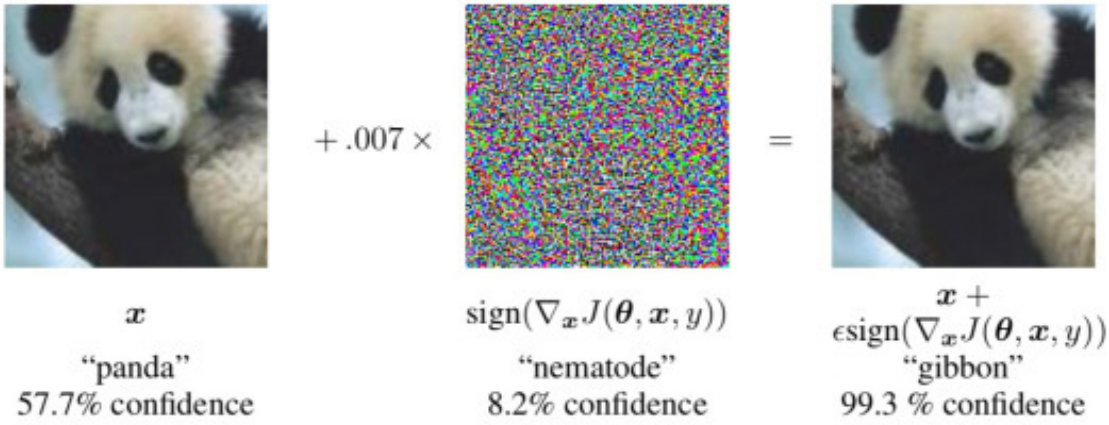


Figure 1: A demonstration of fast adversarial example generation applied to GoogLeNet (Szegedy et al., 2014a) on ImageNet. By adding an imperceptibly small vector whose elements are equal to the sign of the elements of the gradient of the cost function with respect to the input, we can change GoogLeNet's classification of the image. Here our ϵ of .007 corresponds to the magnitude of the smallest bit of an 8 bit image encoding after GoogLeNet's conversion to real numbers.

An intriguing aspect of adversarial examples is that an example generated for one model is often misclassified by other models, even when they have different architecures or were trained on disjoint training sets. Moreover, when these different models misclassify an adversarial example, they often agree with each other on its class. Explanations based on extreme non-linearity and overfitting cannot readily account for this behavior...

But under the linear explanation, adversarial examples occur in broad subspaces – this explains why adversarial examples are abundant and why an example misclassified by one classifier has a fairly high probability of being misclassified by another. It's the direction of perturbation, rather than the specific point in space, that matters most.

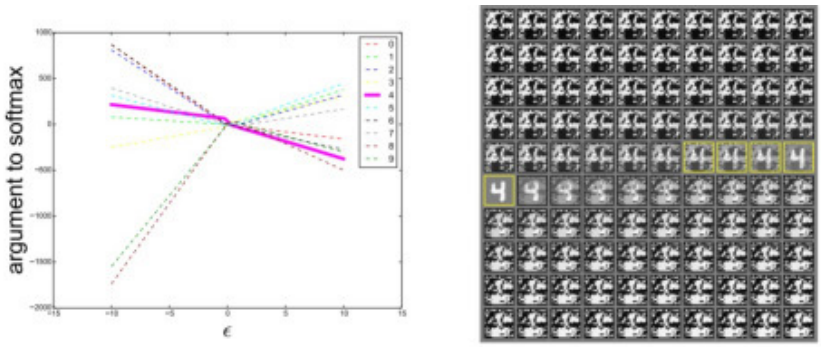


Figure 4: By tracing out different values of ϵ, we can see that adversarial examples occur reliably for almost any sufficiently large value of ϵ provided that we move in the correct direction. Correct classifications occur only on a thin manifold where $\boldsymbol{x}$ occurs in the data. Most of $\mathbb{R}^n$ consists of adversarial examples and *rubbish class examples* (see the appendix). This plot was made from a naively trained maxout network. Left) A plot showing the argument to the softmax layer for each of the 10 MNIST classes as we vary ϵ on a single input example. The correct class is 4. We see that the unnormalized log probabilities for each class are conspicuously piecewise linear with ϵ and that the wrong classifications are stable across a wide region of ϵ values. Moreover, the predictions become very extreme as we increase ϵ enough to move into the regime of rubbish inputs. Right) The inputs used to generate the curve (upper left = negative ϵ, lower right = positive ϵ, yellow boxes indicate correctly classified inputs).

Our explanation suggests a fundamental tension between designing models that are easy to train due to their linearity and designing models that use nonlinear effects to resist adversarial perturbation. In the long run, it may be possible to escape this tradeoff by designing more powerful optimization methods that can successfully train more nonlinear models.

DISTILLATION AS A DEFENSE TO ADVERSARIAL PERTURBATIONS AGAINST DEEP NEURAL NETWORKS

Yesterday we looked at distillation as a way of transferring knowledge from large models to smaller models. In 'Distillation as a defense...,' Papernot et al. show that the distillation technique (training using the probability distribution as the target, not just the argmax class label) can also be used to greatly reduce the vulnerability of networks to adversarial perturbations.

We formulate a new variant of distillation to provide for defense training: instead of transferring knowledge between different architectures, we propose to use the knowledge extracted from a DNN to improve its own resilience to adversarial samples.

With a DNN trained on the MNIST dataset, defensive distillation reduces the success rate of adversarial sample crafting from 95.89% to just 0.45%! For a DNN trained on the CIFAR dataset, the success rate was reduced from 87.89% to 5.11%. In fact, defensive distillation can reduce the sensitivity of a DNN to input perturbations by a whopping factor of 1030. This increases the minimum number of input features that need to be perturbed for adversarial samples to succeed by up to 8x in tests.

Here are some examples from MNIST and CIFAR showing legitimate and adversarial samples:

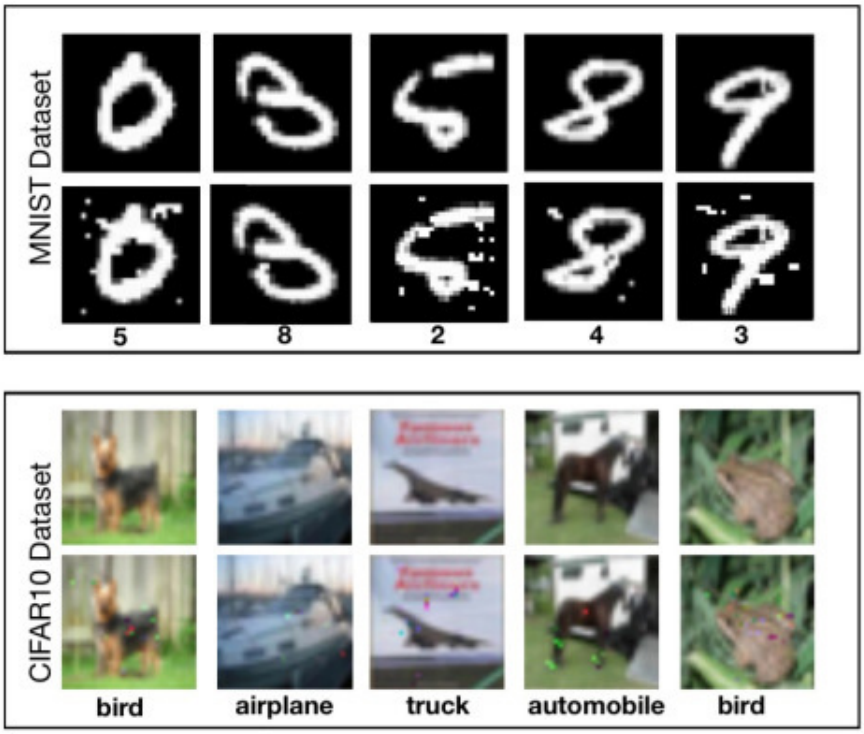


Fig. 2: **Set of legitimate and adversarial samples for two datasets:** For each dataset, a set of legitimate samples, which are correctly classified by DNNs, can be found on the top row while a corresponding set of adversarial samples (crafted using [7]), misclassifed by DNNs, are on the bottom row.

So how and why does defensive distillation work? Consider a general adversarial crafting framework that works by first figuring out the *directions* around a given input sample in which the model learned by a DNN is most sensitive, and then uses this information to select a perturbation among the input dimensions.

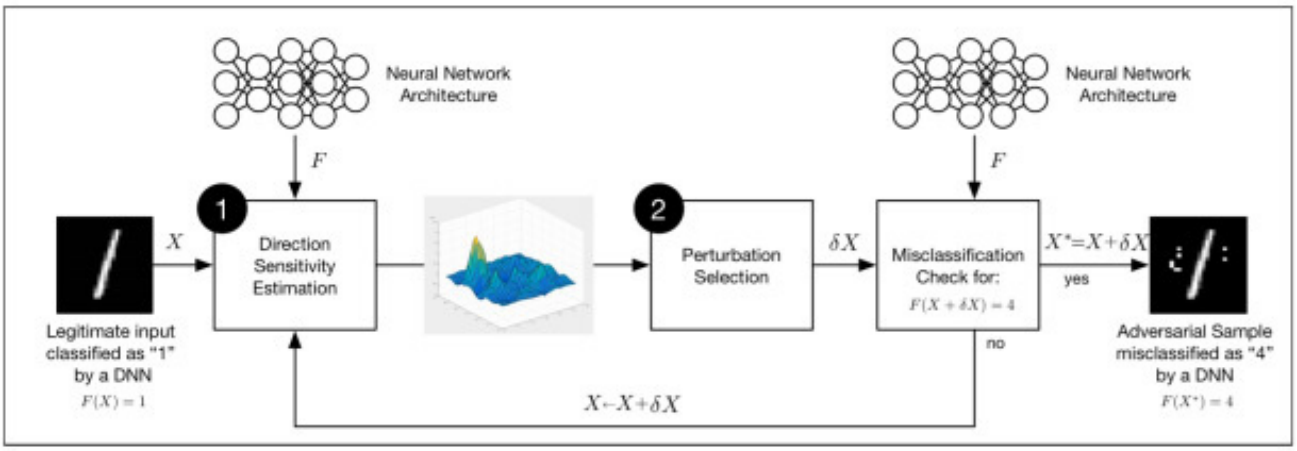


Fig. 3: **Adversarial crafting framework:** Existing algorithms for adversarial sample crafting [7], [9] are a succession of two steps: (1) *direction sensitivity estimation* and (2) *perturbation selection*. Step (1) evaluates the sensitivity of model F at the input point corresponding to sample X. Step (2) uses this knowledge to select a perturbation affecting sample X's classification. If the resulting sample $X + \delta X$ is misclassified by model F in the adversarial target class (here 4) instead of the original class (here 1), an adversarial sample X^* has been found. If not, the steps can be repeated on updated input $X \leftarrow X + \delta X$.

If the direction gradients are steep, we can make a big impact with small perturbations, but if they are shallow this is much harder to achieve. Think about the difference between being on a 'ridge' in the classification space whereby a small move to either side could see you tumbling down the mountain, and being on a plateau where you can freely wander around without much consequence.

To defend against such perturbations, one must therefore reduce these variations around the input, and consequently the amplitude of adversarial gradients. In other words, we must smooth the model learned during training by helping the network generalize better to samples outside of its training dataset.

The 'robustness' of a DNN to adversarial samples is correlated with classifying inputs relatively consistently in the neighbourhood of a given sample.

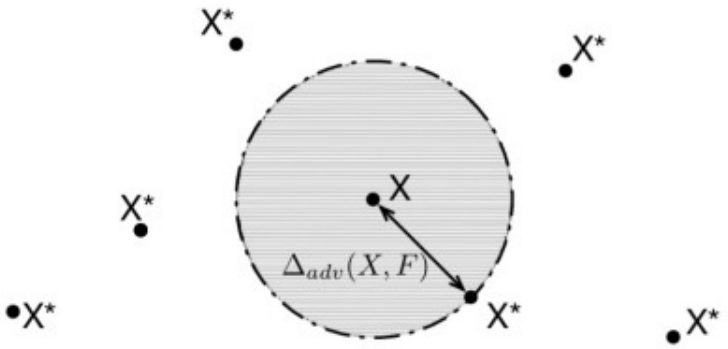


Fig. 4: **Visualizing the hardness metric:** This 2D representation illustrates the hardness metric as the radius of the disc centered at the original sample X and going through the closest adversarial sample X^* among all the possible adversarial samples crafted from sample X. Inside the disc, the class output by the classifier is constant. However, outside the disc, all samples X^* are classified differently than X.

To achieve this smoothing, distillation defense first trains a classification network as normal. Then we take another fresh model instance *with the exact same architecture* (no need to transfer to a smaller model) and train it using the probability vectors learned by the first model.

The main difference between defensive distillation and the original distillation proposed by Hinton et al. is that we keep the same network architecture to train both the original network as well as the distilled network. This difference is justified by our end which is resilience instead of compression.

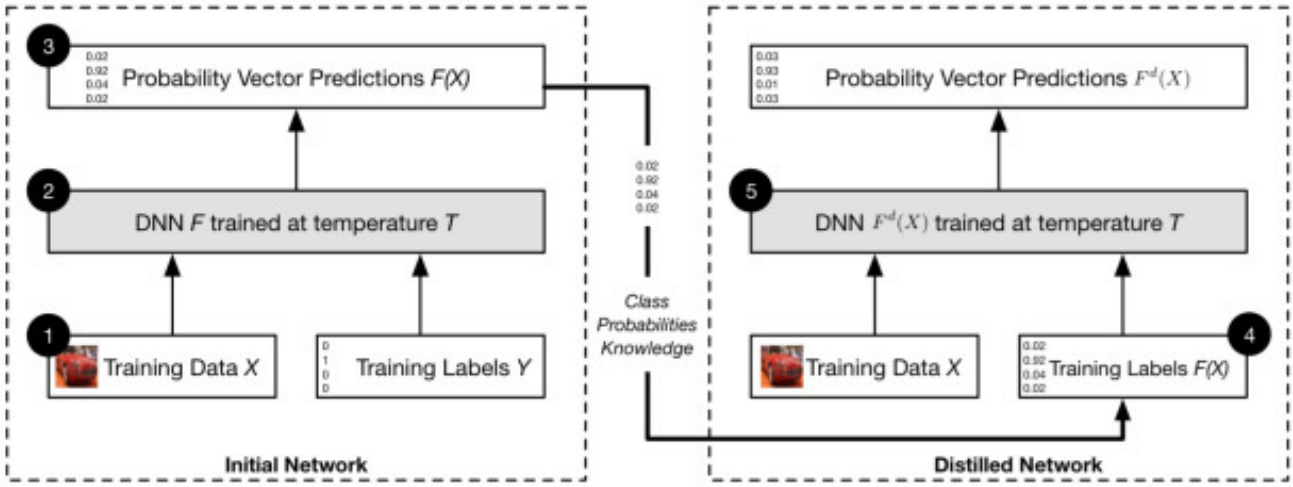
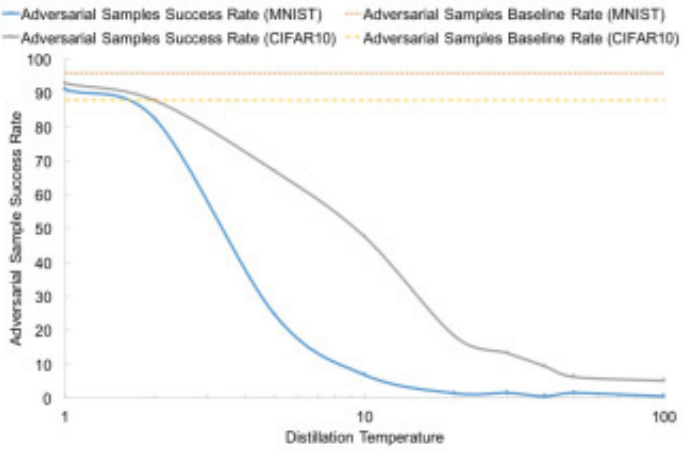


Fig. 5: **An overview of our defense mechanism based on a transfer of knowledge contained in probability vectors through distillation:** We first train an initial network F on data X with a softmax temperature of T. We then use the probability vector $F(X)$, which includes additional knowledge about classes compared to a class label, predicted by network F to train a distilled network F^d at temperature T on the same data X.

Training the network in this way with explicit relative information about classes prevents it from fitting too tightly to the data, and hence contributes to better generalization.

The following figure shows how the distillation *temperature* impacts the model's ability to defend against adversarial samples. Intuitively, the higher the temperature the greater the smoothing, and thus the better the defence.



Distillation Temperature	MNIST Adversarial Samples Success Rate (%)	CIFAR10 Adversarial Samples Success Rate (%)
1	91	92.78
2	82.23	87.67
5	24.67	67
10	6.78	47.56
20	1.34	18.23
30	1.44	13.23
40	0.45	9.34
50	1.45	6.23
100	0.45	5.11
No distillation	95.89	87.89

Fig. 7: **An exploration of the temperature parameter space:** for 900 targets against the MNIST and CIFAR10 based models and several distillation temperatures, we plot the percentage of targets achieved by crafting an adversarial sample while altering at most 112 features. Baselines for models trained without distillation are in dashes. Note the horizontal logarithmic scale.

Distillation has only a small impact on classification accuracy, and *may even improve it!*

We know that many different machine learning models are vulnerable to adversarial attacks, but the defensive distillation defense is only applicable to DNN models that produce an energy-based probability distribution for which a temperature can be defined...

However, note that many machine learning models, unlike DNNs, don't have the model capacity to be able to resist adversarial examples... A defense specialized to DNNs, guaranteed by the universal approximation property to at least be able to represent a function that correctly processes adversarial examples, is thus a significant step towards building machine learning models robust to adversarial samples.

This all sounds quite promising... unfortunately a subsequent paper showed that even defensive distillation is insufficient in mitigating adversarial examples :(, 'Defensive distillation is not robust to adversarial examples.'

In this short paper, we demonstrate that defensive distillation is not effective. We show that, with a slight modification to a standard attack, one can find adversarial examples on defensively distilled networks. We demonstrate the attack on the MNIST digit recognition task. Distillation prevents existing techniques from finding adversarial examples by increasing the magnitude of the inputs to the softmax layer. This makes an unmodified attack fail. We show that if we artificially reduce the magnitude of the input to the softmax function, and make two other minor changes, the attack succeeds. Our attack achieves successful targeted misclassification on 96.4% of images by changing on average 4.7% of pixels.

Damn!

VULNERABILITY OF DEEP REINFORCEMENT LEARNING TO POLICY INDUCTION ATTACKS

If you weren't there already, this is where we get to the 'Oh *#@!' moment! We've seen that classifiers can be fooled, but this paper and the next one show us that deep reinforcement learning networks (e.g. DQNs) are also vulnerable to adversarial attack. The attack is demonstrated on Atari games (what else!), but the broader implications are sobering:

The reliance of RL on interactions with the environment gives rise to an inherent vulnerability which makes the process of learning susceptible to perturbation as a result of changes in the observable environment. Exploiting this vulnerability provides adversaries with the means to disrupt or change control policies, leading to unintended and potentially harmful actions. For instance, manipulation of the obstacle avoidance and navigation policies learned by autonomous Unmanned Aerial Vehicles (UAV) enables the adversary to use such systems as kinetic weapons by inducing actions that lead to intentional collisions.

Fortunately, we've already seen many of the building blocks needed to craft the attack, so we can describe it quite succinctly. The goal of the attacker is to fool a DQN into taking an action (inducing an arbitrary policy) chosen by the attacker. The assumed threat model is similar to the 'black-box' model we saw earlier in this post: the attacker has no visibility of the insides of the DQN, and does not know its reward function. However, the attacker can see the same environmental inputs that the target DQN sees, and it can observe the actions taken by the DQN and hence estimate the reward function.

The first step is to us the 'Practical black-box attack...' technique to train a substitute DQN that matches the policies chosen by the target. Following the black-box playbook we now craft adversarial *inputs* (instead of images) that trigger an incorrect choice of optimal action...

If the attacker is capable of crafting adversarial inputs s'tand s't+1 such that the value of [the training function] is minimized for a specific action a', then the policy learned by the DQN at this time-step is optimized for suggesting a' as the optimal action given the state st.

(An example of adversarial inputs might be manipulating some of the screen input pixels in an Atari game).

At this point we have a DQN which has learned an adversarial policy. The next step in the playbook is to find a way to *transfer* this learned adversarial policy to the target network. This is done in an *exploitation cycle:*

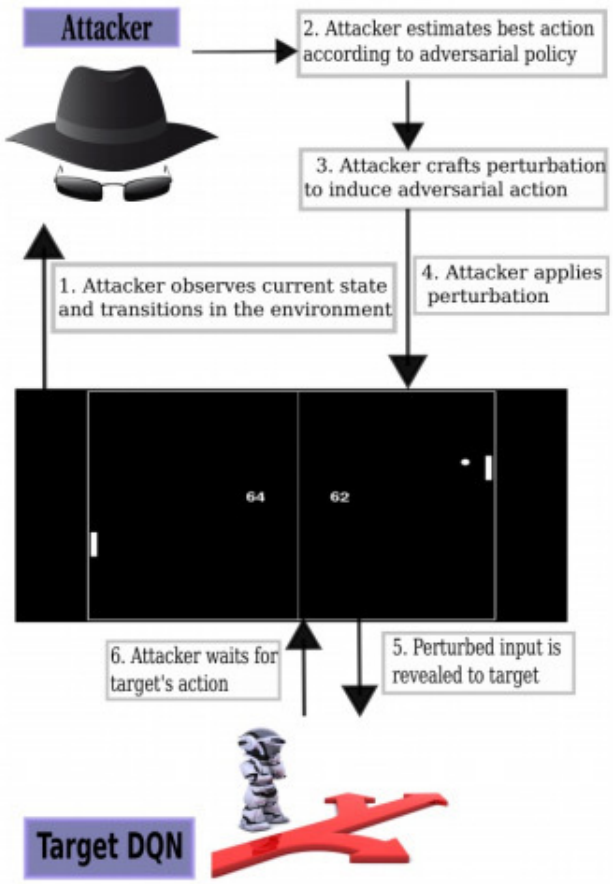


Fig. 2: Exploitation cycle of policy induction attack

The first question we need to answer therefore, is 'is it possible to generate adversarial examples for DQNs?' Fig. 4 below shows that yes, this is indeed possible (game of Atari Pong, using both the Fast Gradient Sign and Jacobian Saliency Map Algorithm approaches to generate adversarial perturbations)

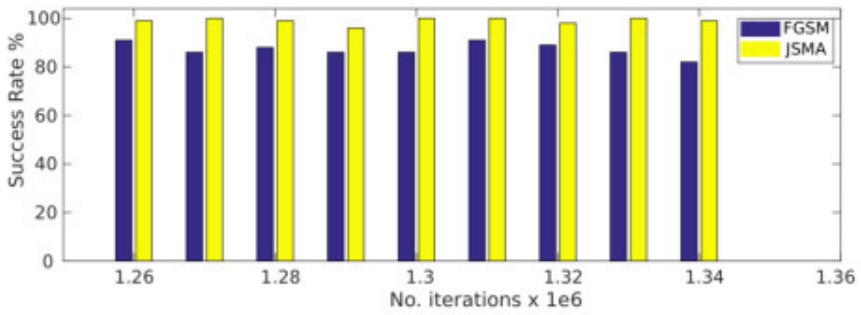


Fig. 4: Success rate of crafting adversarial examples for DQN

The next question we have to answer, is whether or not these adversarial examples can be transferred. The answer again is yes, with high success rate:

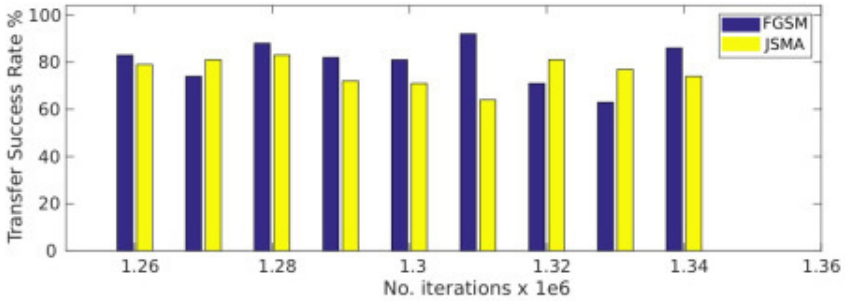


Fig. 5: Transferability of adversarial examples in DQN

Our final experiment tests the performance of our proposed exploitation mechanism. In this experiment, we consider an adversary whose reward value is the exact opposite of the game score, meaning that it aims to devise a policy that maximizes the number of lost games. To obtain this policy, we trained an adversarial DQN on the game, whose reward value was the negative of the value obtained from target DQN's reward function...

A picture is worth a thousand words here:

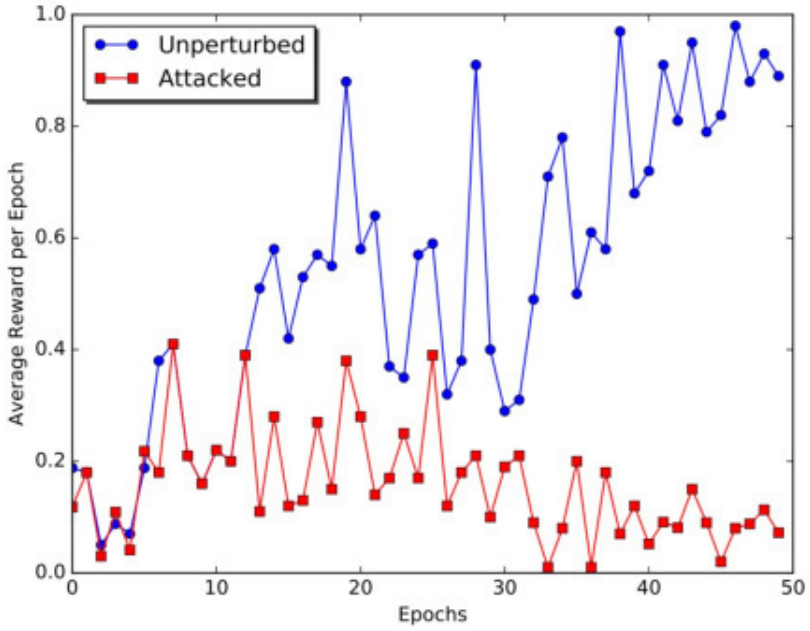


Fig. 6: Comparison of rewards between unperturbed and attacked DQNs

Since all known counter-measures have been shown not to be sufficient,

... it is hence concluded that the current state of the art in countering adverse examples and their exploitation is incapable of providing a concrete defense against such exploitations.

ADVERSARIAL ATTACKS ON NEURAL NETWORK POLICIES

Almost in parallel to the previous paper, Huang et al. published this work which also shows that reinforcement learning networks are vulnerable to adversarial attacks. They demonstrate this across *four* different Atari games (Chopper Command, Pong, Seaquest, and Space Invaders) using white-box attacks. They also show that the attacks succeed across a range of deep reinforcement learning algorithms (DQN, TRPO, and A3C). Policies trained with TRPO and A3C are more resistant, but not safe from the attack.

Then the authors demonstrate transfer capabilities using black-box attacks too:

We observe that the cross-dataset transferability property also holds in reinforcement learning applications, in the sense that an adversarial example designed to interfere with the operation of one policy interferes with the operation of another policy, so long as both policies have been trained to solve the same task. Specifically, we observe that adversarial examples transfer between models trained using different trajectory rollouts and between models trained with different training algorithms.

Combine this with the lessons we learned above in 'Adversarial examples in the physical world,' and as the authors point out, things could get very interesting indeed!

Our experiments show it is fairly easy to confuse such policies with computationally-efficient adversarial examples, even in black-box scenarios. Based on 'Adversarial examples in the physical world', it is possible that these adversarial perturbations could be applied to objects in the real world, for example adding strategically-placed paint to the surface of a road to confuse an autonomous car's lane-following policy

“ it is possible that these adversarial perturbations could be applied to objects in the real world, for example adding strategically-placed paint to the surface of a road to confuse an autonomous car's lane-following policy.

THOU SHALT NOT DEPEND ON ME

ANALYSING THE USE OF OUTDATED JAVASCRIPT LIBRARIES ON THE WEB

Just based on the paper title alone, if you had to guess what the situation is with outdated JavaScript libraries on the web, you'd probably guess it was pretty bad.

Lauinger et al., NDSS 2017

It turns out it's very bad indeed, and we've created a huge mess with nowhere near enough attention being paid to the issue. The first step towards better solutions is recognising that we have a problem, and Lauinger et al., do a tremendous job in that regard.

In this paper, we conduct the first comprehensive study of client-side JavaScript library usage and the resulting security implications across the Web. Using data from over 133K websites, we show that 37% of them include at least one library with a known vulnerability; the time lag behind the newest release of a library is measured in the order of years.

For example, 36.7% of jQuery includes, 40.1% of Angular, and an astonishing 86.6% of Handlebars includes use a vulnerable version. Those are headline grabbing numbers, but when you go deeper it turns out there's no quick fix in sight because the root causes are systemic in the JavaScript ecosystem:

Perhaps our most sobering finding is practical evidence that the JavaScript library ecosystem is complex, unorganised, and quite "ad hoc" with respect to security. There are no reliable vulnerability databases,

Using data from over 133K websites, we show that 37% of them include at least one library with a known vulnerability.

no security mailing lists maintained by library vendors, few or no details on security issues in release notes, and often, it is difficult to determine which versions of a library are affected by a specific reported vulnerability.

Let's briefly look at how the authors collected their data before diving deeper into what the results themselves tell us.

DATA GATHERING METHODOLOGY

The team crawled the Alexa Top 75K websites (ALEXA) and also a random sample of 75K websites drawn from the **.com** domain (COM). This enables a comparison of JavaScript usage across popular and unpopular websites.

Figuring out which JavaScript libraries were actually in use, and their versions, took quite a bit of work. As did figuring out which versions contain vulnerabilities. Details of the tools and techniques used for this can be found in the paper, in brief it involved:

- Manually constructing a catalogue of all releases versions of the 72 most popular open source libraries (using popularity statistics from Bower and Wappalyzer).
- Using static and dynamic analysis techniques to cope with the fact that developers often reformat, restructure, or append code making it difficult to detect library usage in the wild
- Implementing an in-browser causality tracker to understand *why* specific libraries are loaded by a given site.

VULNERABILITIES

The last step towards building our catalogue is aggregating vulnerability information for our 72 JavaScript libraries. Unfortunately, there is no centralised database of vulnerabilities in JavaScript libraries; instead, we manually compile vulnerability information from the Open Source Vulnerability Database (OSVDB), the National Vulnerability Database (NVD), public bug trackers, GitHub comments, blog posts, and the vulnerabilities detected by Retire.js. Overall, we are able to obtain systematically documented details of vulnerabilities for 11 of the JavaScript libraries in our catalogue.

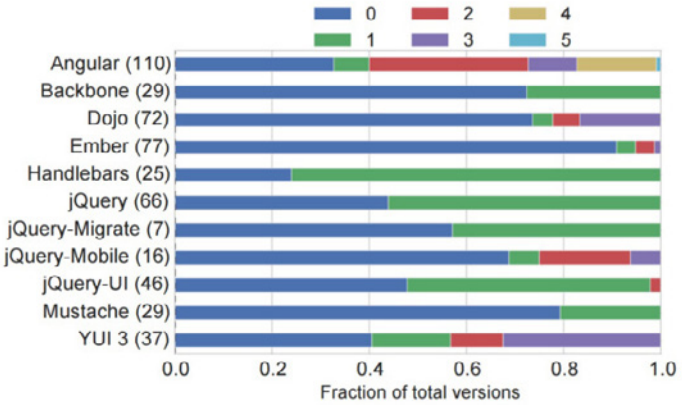


Fig. 1. Fraction of library versions with i distinct known vulnerabilities each (represented by colours), out of the total library versions in parentheses. Angular `1.2.0` has 5 known vulnerabilities and there are 110 versions overall.

CAUSALITY TREES

To figure out why a certain library is being loaded, the authors develop a *causality*

tree chrome extension. Nodes in the tree are snapshots of elements in the DOM at a specific point in time, and edges denote "created by" relationships.

For example:

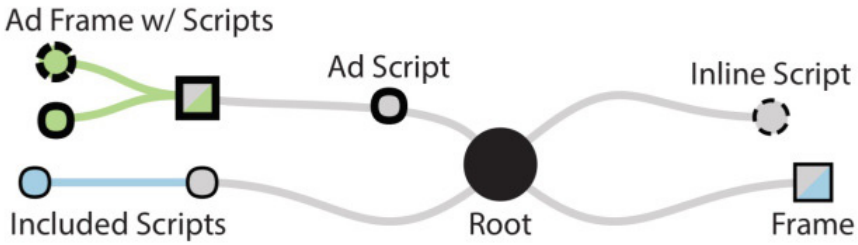


Fig. 2. Example causality tree.

and

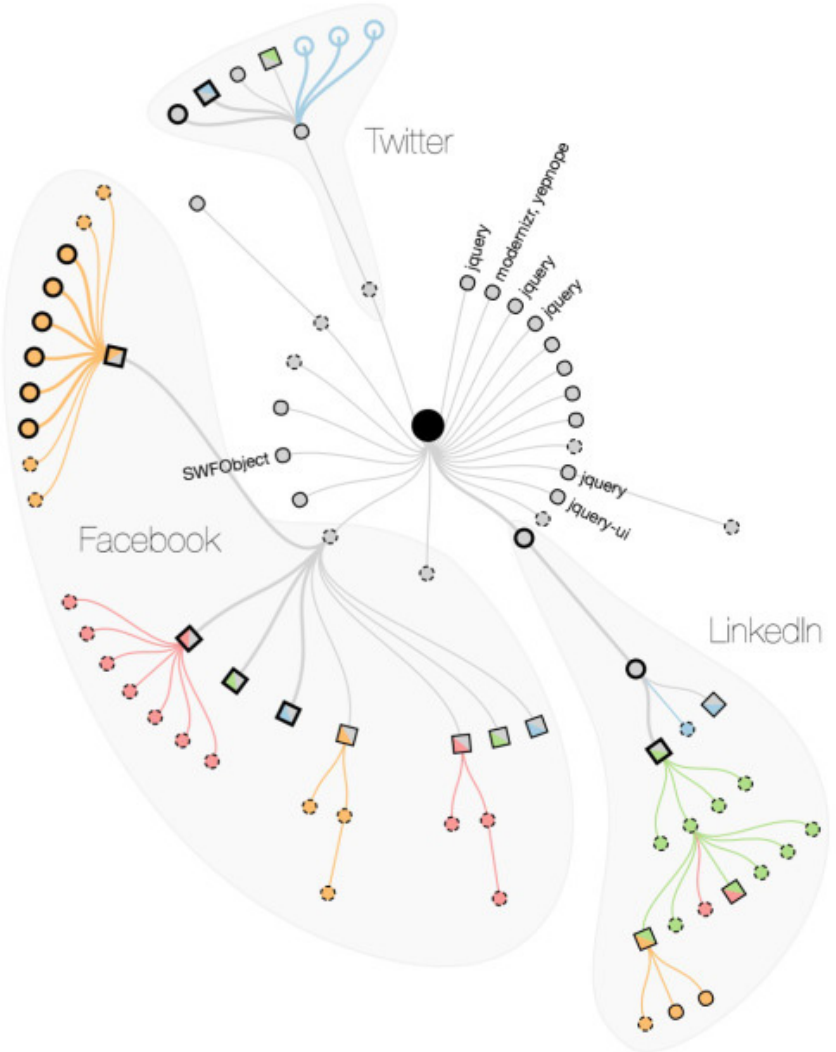


Fig. 12. Causality tree for `mercantil.com` with multiple jQuery inclusions, two libraries concatenated into one file (Modernizr and Yepnope), and complex element relationships in social media widgets (Twitter, Facebook and LinkedIn).

The median causality tree in ALEXA contains 133 nodes, and the median depth is 4 inclusions.

JAVASCRIPT LIBRARY MARKET SHARE

jQuery remains by far the most popular library, found on 84.5% of ALEXA sites. Note also SWFObject (Adobe Flash) still used on 10.7% of ALEXA sites despite being discontinued in 2013. (Table I)

When externally loaded, scripts are mostly loaded from CDNs (note also the domain parking sites popping up in the long tail of the COM sites): Table II

Overall though, there seems to be a pretty even split between internally hosted and CDN-delivered script libraries: Table IV

DISTRIBUTION OF VULNERABLE LIBRARIES

37.8% of ALEXA sites use at least one library version known to the authors to be vulnerable.

Highly-ranked websites tend to be less likely to include vulnerable libraries, but they are also less likely to include any detected library at all. Towards the lower ranks, both curves increase at a similar pace until they stabilise. While only 21 % of the Top 100 websites use a known vulnerable library, this percentage increases to 32.2 % in the Top 1 k before it stabilises in the Top 5 k and remains around the overall average of 37.8 % for all 75 k websites.

TABLE I. THE 30 MOST FREQUENT LIBRARIES IN OUR ALEXA WEB CRAWL (OUT OF 72 SUPPORTED LIBRARIES). *DYNAMIC METHOD DETECTS ● ALL KNOWN VERSIONS, ◗ RECENT VERSIONS, ❍ ONLY THE LIBRARY'S PRESENCE; OTHERWISE DETECTION BY STATIC METHOD ONLY.

Library	Versions	*Dyn.	Bower Rank	Wapp %	Use on Crawled Sites ALEXA	COM
jQuery	66	●	1	42 %	84.5 %	62.6 %
jQuery-UI	46	●	13	7 %	24.7 %	8.6 %
Modernizr	24	●	18	10 %	22.1 %	9.3 %
Bootstrap	32	◗	3		13.7 %	4.8 %
Yepnope	10	❍		6 %	13.6 %	5.3 %
jQuery-Migrate	7	●			11.3 %	10.7 %
SWFObject	2	◗			10.7 %	5.2 %
Underscore	61	●	12	3 %	5.8 %	2.4 %
jQuery-Tools	8	◗			4.0 %	1.2 %
Flexslider	11	❍			3.6 %	1.6 %
Moment	54	●	6		3.5 %	1.4 %
RequireJS	62	●			3.4 %	2.3 %
jQuery-Form	14				2.7 %	3.4 %
Backbone	29	●		2 %	2.7 %	1.6 %
Angular	110	◗	2		2.5 %	1.6 %
LoDash	77	◗	26		2.4 %	2.5 %
jQuery-Fancybox	10				2.3 %	1.4 %
GreenSock GSAP	45				2.3 %	2.8 %
Handlebars	25	●			2.2 %	0.5 %
Prototype	5	●			2.2 %	1.3 %
Hammer.js	26	◗			1.9 %	1.1 %
FastClick	33	❍	23		1.7 %	0.4 %
WebFont Loader	100				1.7 %	1.3 %
MooTools	27	●		3 %	1.6 %	1.4 %
Isotope	38				1.6 %	1.8 %
jQuery-Cookie	8				1.4 %	0.2 %
Knockout	21	●			1.4 %	0.3 %
jQuery-Mobile	16	◗			1.4 %	0.8 %
Mustache	29	●			1.3 %	1.0 %
jQuery-Validation	13				1.1 %	0.6 %

TABLE III. DETECTED JS LIBRARIES: TOP 10 MARKET SHARE. SORTED BY ALEXA; DATA OMITTED FOR HOSTS NOT PART OF THE TOP 10.

Hostname	ALEXA	COM
ajax.googleapis.com	16.7 %	13.3 %
code.jquery.com	3.3 %	4.0 %
static.hugedomains.com		4.0 %
static.parastorage.com		3.1 %
img.sedoparking.com		2.2 %
cdnjs.cloudflare.com	2.0 %	1.3 %
ak2.imgaft.com		2.0 %
img4.wsimg.com		2.0 %
static.squarespace.com		1.9 %
s0.2mdn.net	1.4 %	
ad.turn.com	1.4 %	
ivid-cdn.adhigh.net	1.1 %	
cdn2.editmysite.com		1.1 %
maxcdn.bootstrapcdn.com	0.9 %	
cdn10.trafficjunky.net	0.4 %	
netdna.bootstrapcdn.com	0.4 %	
ajax.aspnetcdn.com	0.3 %	

TABLE IV. INCLUSION TYPES OF DETECTED LIBRARIES: LOADED FROM INLINE/INTERNAL/EXTERNAL SCRIPT & INCLUSIONS (OF ANY TYPE) DUE TO AD/TRACKER/WIDGET CODE. GREYED OUT IF <100 INCLUSIONS; OMITTED IF <10. LIBRARIES COUNTED AT MOST ONCE PER SITE.

	ALEXA				COM			
	Included From			Ad/Tr./	Included From			Ad/Tr./
Library	Inl.	Int.	Ext.	Widget	Inl.	Int.	Ext.	Widget
jQuery	2.8	**58.6**	38.7	4.1	2.8	29.0	**68.1**	25.4
Modernizr	3.1	**65.4**	31.5	11.9	1.4	38.2	**60.4**	3.2
Bootstrap	1.8	**70.4**	27.7	0.3	0.5	49.6	**49.8**	3.3
Yepnope	2.3	**68.7**	29.0	10.3	1.7	48.0	**50.3**	4.3
SWFObject	3.2	45.9	**50.9**	35.6	1.4	24.7	**73.9**	7.7
Angular	2.4	**56.5**	41.1	3.9		6.0	**93.9**	5.2
YUI 3		45.6	**54.2**	20.4	6.6	1.9	**91.5**	2.3
Ember	10.5	14.6	**74.8**	35.3			**95.7**	37.3
Dojo	33.3	**52.9**	13.8		40.5		52.7	

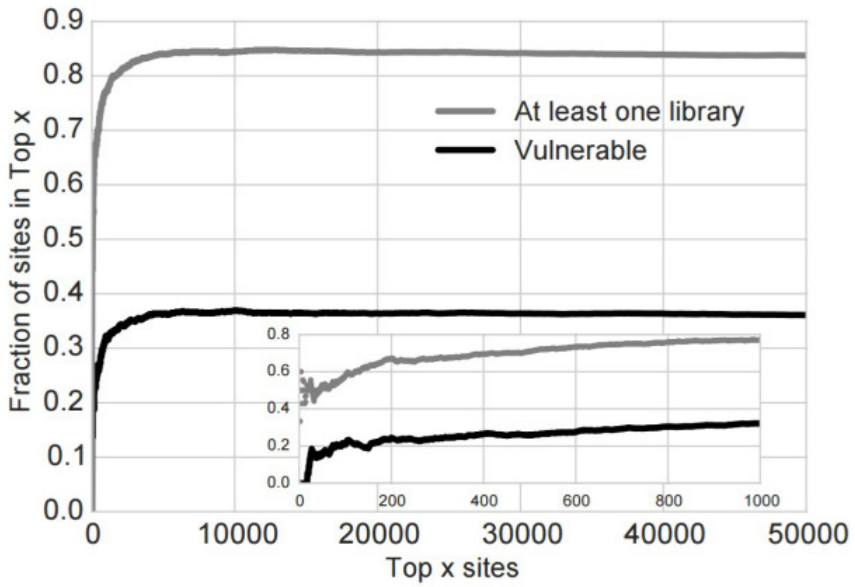


Fig. 9. Fraction of websites with detected vulnerable inclusions in the ALEXA Top x, and fraction of websites with at least one detected library for comparison. Vulnerable libraries can be found on 26 % of Top 500 websites.

37.4% of the COM sites use at least one vulnerable library. Within the ALEXA grouping, *financial and government sites are the worst*, with 52% and 50% of sites containing vulnerable libraries respectively.

The following table shows the percentage of vulnerable copies in the wild for jQuery, jQ-UI, Angular, Handlebars, and YUI 3.

TABLE V. VULNERABLE FRACTION OF INCLUSIONS PER LIBRARY, COUNTING AT MOST ONE LIBRARY-VERSION PAIR PER SITE. CONFIGURATIONS WITH LESS THAN 100 VULNERABLE INCLUSIONS GRAYED OUT; OMITTED IF LESS THAN 10 TOTAL INCLUSIONS AFTER FILTER.

(a) ALEXA

Inclusion Filter	jQuery	jQ-UI	Angular	Handlebars	YUI 3
All Inclusions	36.7 %	33.7 %	40.1 %	86.6 %	87.3 %
Internal	38.1 %	33.0 %	37.1 %	84.6 %	92.8 %
External	34.8 %	35.5 %	48.2 %	88.6 %	83.8 %
Inline	54.8 %	30.7 %	20.0 %	100.0 %	-
Internal Parent	37.1 %	33.7 %	40.6 %	85.3 %	91.3 %
External Parent	32.6 %	33.8 %	41.8 %	96.4 %	92.2 %
Inline Parent	47.6 %	35.2 %	25.6 %	83.7 %	79.6 %
Direct Incl. in Root	36.4 %	33.5 %	40.1 %	85.2 %	90.4 %
Indirect Inclusion	41.0 %	35.6 %	43.2 %	92.3 %	87.1 %
WordPress	29.2 %	14.4 %	23.5 %	77.3 %	-
Non-WordPress	36.8 %	34.2 %	40.7 %	86.8 %	87.3 %
Ad/Widget/Tracker	38.1 %	39.8 %	23.5 %	96.9 %	98.2 %
No Ad/Widget/Tracker	36.7 %	33.6 %	40.8 %	86.5 %	84.1 %

(b) COM

Inclusion Filter	jQuery	jQ-UI	Angular	Handlebars	YUI 3
All Inclusions	55.4 %	37.3 %	38.7 %	87.5 %	13.7 %
Internal	41.6 %	28.1 %	45.8 %	100.0 %	68.8 %
External	62.7 %	42.7 %	38.4 %	85.4 %	12.6 %
Inline	89.9 %	25.6 %	-	-	-
Internal Parent	59.7 %	37.0 %	78.2 %	94.2 %	47.9 %
External Parent	45.9 %	38.0 %	20.7 %	84.1 %	68.9 %
Inline Parent	79.8 %	48.9 %	-	-	1.0 %
Direct Incl. in Root	42.6 %	36.2 %	41.3 %	88.7 %	50.4 %
Indirect Inclusion	77.5 %	44.5 %	30.8 %	86.4 %	7.5 %
WordPress	41.6 %	20.4 %	25.0 %	70.3 %	-
Non-WordPress	55.6 %	38.6 %	38.9 %	90.7 %	13.7 %
Ad/Widget/Tracker	89.0 %	31.6 %	19.2 %	-	55.0 %
No Ad/Widget/Tracker	45.5 %	37.3 %	39.6 %	87.3 %	12.7 %

In ALEXA, 36.7% of jQuery inclusions are known vulnerable, when at most one inclusion of a specific library version is counted per site. Angular has 40.1% vulnerable inclusions, Handlebars has 86.6%, and YUI 3 has 87.3% (it is not maintained any more). These numbers illustrate that inclusions of known vulnerable versions can make up even a majority of all inclusions of a library.

Many libraries it turns out are not *directly* included by the site, but are pulled in by other libraries that are. "*Library inclusions by ad, widget, or tracker code appear to be more vulnerable than unrelated inclusions.*"

Another interesting analysis is the *age* of the included libraries – the data clearly shows that the majority of web sites use library versions released a long time ago, suggesting that developers rarely update their library dependencies once they have deployed a site. 61.7% of ALEXA sites are at least one patch version behind on one of their included libraries, and the median ALEXA site uses a version released 1,177 days before the newest release of the library. Literally *years* out of date.

DUPLICATE INCLUSIONS

If you like a little non-determinism in your web app (I find it always make debugging *much* more exciting), then another interesting find is that many sites include the same libraries (and multiple versions thereof) many times over!

We discuss some examples using jQuery as a case study. About 20.7 % of the websites including jQuery in ALEXA (17.2 % in COM) do so two or more times. While it may be necessary to include a library multiple times within different documents from different origins, 4.2 % of websites using jQuery in ALEXA include the same version of the library two or more times into the same document (5.1 % in COM), and 10.9 % (5.7 %) include two or more different versions of jQuery into the same document. Since jQuery registers itself as a window-global variable, unless special steps are taken only the last loaded and executed instance can be used by client code. For asynchronously included instances, it may even be difficult to predict which version will prevail in the end.

WHAT CAN BE DONE?

So where does all this leave us?

From a remediation perspective, the picture painted by our data is bleak. We observe that only very small fraction of potentially vulnerable sites (2.8 % in ALEXA, 1.6 % in COM) could become free of vulnerabilities by applying patch-level updates, i.e., an update of the least significant version component, such as from 1.2.3 to 1.2.4, which would generally be expected to be backwards compatible. The vast majority of sites would need to install at least one library with a more recent major or minor version, which might necessitate additional code changes due to incompatibilities.

Version aliasing could potentially help (specifying only a library prefix, and allowing the CDN to return the latest version), but only a tiny percentage of sites use it (would *you* trust the developers of those libraries not to break your site, completely outside of your control?). Note that:

Google recently discontinued this service, citing caching issues and "lack of compatibility between even minor versions."

We need proper dependency management which makes it clear which versions of libraries are being used, coupled with knowledge within the supply chain of vulnerabilities. "*This functionality would ideally be integrated into the dependency management system of the platform so that a warning can be shown each time a developer includes a known vulnerable component from the central repository.*"

Of course, that can only work if we have some way of figuring out which libraries are vulnerable in the first place. The state of the practice here is pretty damning :

Unfortunately, security does not appear to be a priority in the JavaScript library ecosystem. Popular vulnerability databases contain nearly no entries regarding JavaScript libraries. During this entire work, we did not encounter a single popular library that had a dedicated mailing list for security announcements (in fact, most libraries we investigated did not have a mailing list for announcements at all). Furthermore, only a few JavaScript library developers provide a dedicated email address where users can submit vulnerability reports....

Consider jQuery, one of the most widely used libraries:

Although jQuery is an immensely popular library, the fact that searching for "security" or "vulnerability" in the official learning centre returns "Apologies, but nothing matched your search criteria" is an excellent summary of the state of JavaScript library security on the Internet, circa August 2016

Since we also know that many libraries are only indirectly loaded by web sites, and are brought in through third-party components such as advertising, tracking, and social media code, even web developers trying to stay on top of the situation may be unaware that they are indirectly introducing vulnerable code into their websites.

REDUNDANCY DOES NOT IMPLY FAULT TOLERANCE

ANALYSIS OF DISTRIBUTED STORAGE REACTIONS TO SINGLE ERRORS AND CORRUPTIONS

Ganesan et al., FAST 2017

It's a tough life being the developer of a distributed datastore. Thanks to the wonderful work of Kyle Kingsbury (aka, @aphyr) and his efforts on Jepsen.io, awareness of data loss and related issues in the face of network delays and partitions is way higher than it used to be. Under the Jepsen tests, every single system tested other than ZooKeeper exhibited problems of some kind. But many teams are stepping up, commissioning tests, and generally working hard to fix the issues even if they're not all there yet.

So I'm sure they'll all be delighted to know that there's a new horror in town. Network partitions aren't the only kind of nasty that distributed datastores can be subjected too. File system / storage devices can exhibit corruptions and errors. And just like network partitions, these aren't as rare as the designers of datastores would like to have you believe. So what happens if you take eight popular distributed stores, and subject them not to a file-system related stress test, but just to one single itsy-bitsy local filesystem fault? Carnage!

The most important overarching lesson from our study is this: a single file-system fault can induce catastrophic outcomes in most modern distributed storage systems. Despite the presence of checksums, redundancy, and other resiliency methods prevalent in distributed storage, a single untimely file-system fault can lead to data loss, corruption, unavailability, and, in some cases, the spread of corruption to other intact replicas.

And this time around, not even ZooKeeper escaped unscathed.

FILE SYSTEM FAULTS

File systems can encounter faults for a variety of underlying causes including media errors, mechanical and electrical problems in the disk, bugs in firmware, and problems in the bus controller. Sometimes, corruptions can arise due to software bugs in other parts of the operating system, device drivers, and sometimes even due to bugs in file systems themselves. Due to these reasons, two problems arise for file systems: block errors, where certain blocks are inaccessible (also called latent sector errors)

and block corruptions, where certain blocks do not contain the expected data.

A study of 1 million disk drives over a period of 32 months showed that 8.5% of near-line disks, and 1.9% of enterprise class disks developed one or more latent sector errors. Flash-based SSDs show similar error rates. Block corruptions (for example caused by bit-rot that goes undetected by the in-disk EEC) may corrupt blocks in ways not detectable by the disk itself. File systems in many cases silently return these corrupted blocks to applications. A study of 1.53 million drives over 41 months showed more than 400,000 blocks with checksum mismatches. Anecdotal evidence has also shown the prevalence of storage errors and corruptions (see e.g. 'Data corruption is worse than you know').

Given the frequency of storage corruptions and errors, there is a non-negligible probability for file systems to encounter such faults.

Type of Fault		Op	Example Causes
Corruption	zeros, junk	Read	misdirected and lost writes in *ext* and *XFS*
Error	I/O error (EIO)	Read	latent sector errors in all file systems, disk corruptions in *ZFS*, *btrfs*
		Write	file system mounted read-only, on-disk corruptions in *btrfs*
	Space error (ENOSPC, EDQUOT)	Write	disk full, quota exceeded in all file systems

Table 1: **Possible Faults and Example Causes.** *The table shows file-systems faults captured by our model and example root causes that lead to a particular fault during read and write operations.*

In many cases file systems simply pass faults (errors or corrupted data) onto applications as-is, in a few other cases the file system may transform the fault into a different one. In either case, throughout the paper these are referred to as *file-system faults.*

Given that local file systems can return corrupted data or errors, the responsibility of data integrity and proper error handling falls to applications, as they care about safely storing and managing critical user data.... The behaviour of modern distributed storage systems in response to file-system faults is critical and strongly affects cloud-based services.

> **a single untimely file-system fault can lead to data loss, corruption, unavailability, and, in some cases, the spread of corruption to other intact replicas.**

ERRFS – THE TEST HARNESS THAT STRIKES FEAR IN THE HEART OF DISTRIBUTED STORAGE SYSTEMS

CORDS is a fault-injection system consisting of *errfs*, a FUSE file system, and *errbench*, a set of workloads and a behaviour inference script for each system under test.

Once a system has been initialized to a known state, the application (distributed datastore) is configure to run on top of errfs by specifying its mount point as the data-directory of the application. All reads and writes then flow through errfs, which can inject faults. Errfs can inject two different types of corruptions (zeros or junk) and three different types of errors (EIO on reads, EIO on writes, and ENOSPC/EDQUOT on writes that require additional space. These

The fault injection is about as kind as it's possible to be:

> *... our model considers injecting example a single fault to a single file-system block in a single node at a time. While correlated file-system faults are interesting, we focus on the most basic case of injecting a single fault in a single node because our fault model intends to give maximum recovery leeway for applications.*

Is *errfs* realistic?

> *All the bugs that we find can occur on XFS and all ext file systems including ext4, the default Linux file system. Given that these file systems are commonly used as local file systems in replicas of large distributed storage deployments and recommended by developers, our findings have important implications for such real-world deployments.*

Given the single fault, the authors test for the following expected behaviours:

- committed data should not be lost
- queries should not silently return corrupted data
- the cluster should be available for reads and writes
- queries should not fail after retries

Using CORDS, the authors tested Redis, ZooKeeper, Cassandra, Kafka, RethinkDB, MongoDB, LogCabin, and CockroachDB. All systems were configured with a cluster of three nodes, and a replication factor of 3.

OK, SO THIS ISN'T GOOD....

> *We find that these systems can silently return corrupted data to users, lose data, propagate corrupted data to intact replicas become unavailable, or return an unexpected error on queries. For example, a single write error during log initialization can cause write unavailability in ZooKeeper. Similarly, corrupted data in one node in Redis and Cassandra can be propagated to other intact replicas. In Kafka and RethinkDB, corruption in one node can cause a user-visible data loss.*

Section 4 of the paper contains a system-by-system breakdown of the problems that the testing uncovered. The results are sum-

marised in this monster table, but you'll be better off reading the detailed description for any particular system of interest.

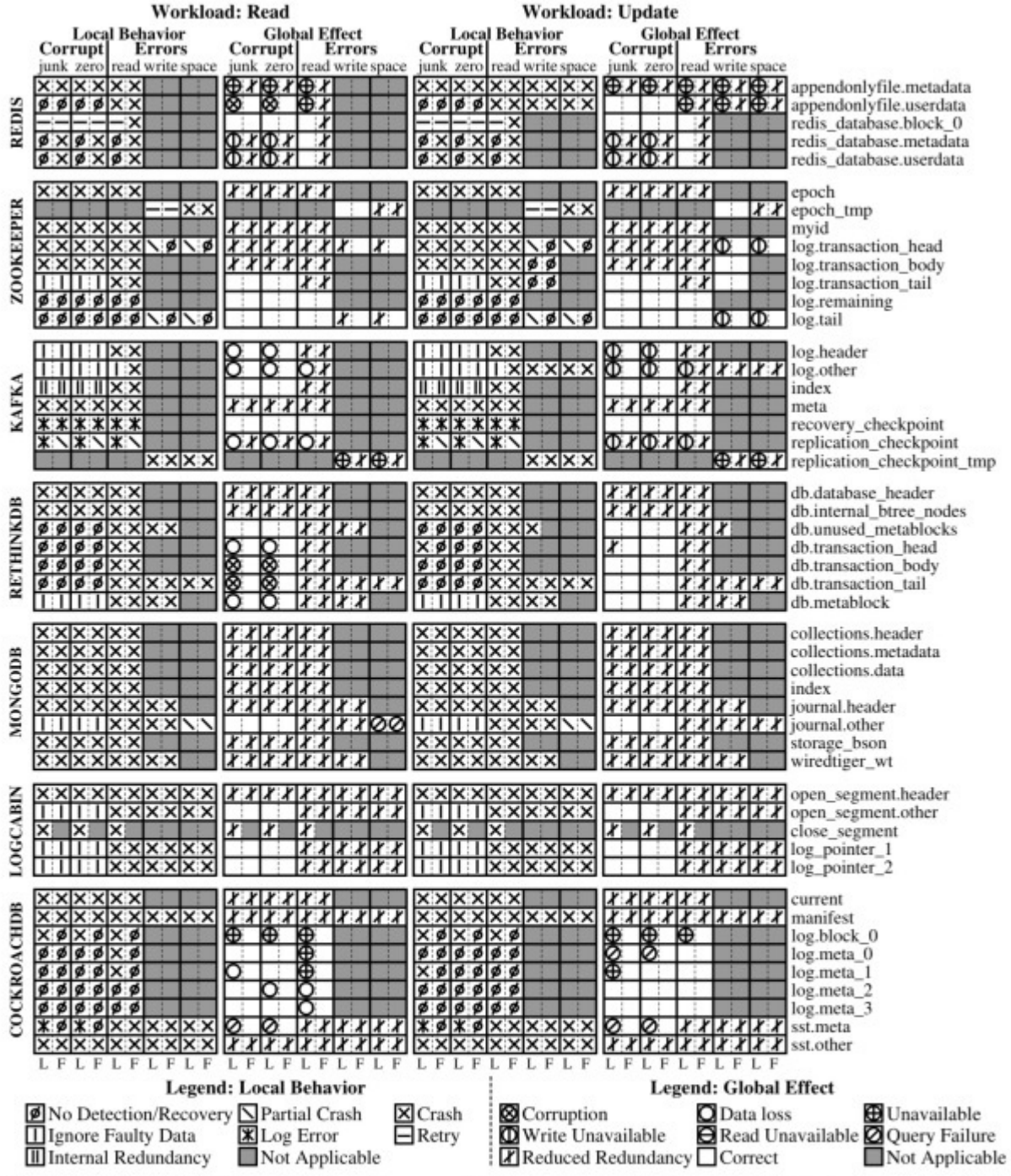
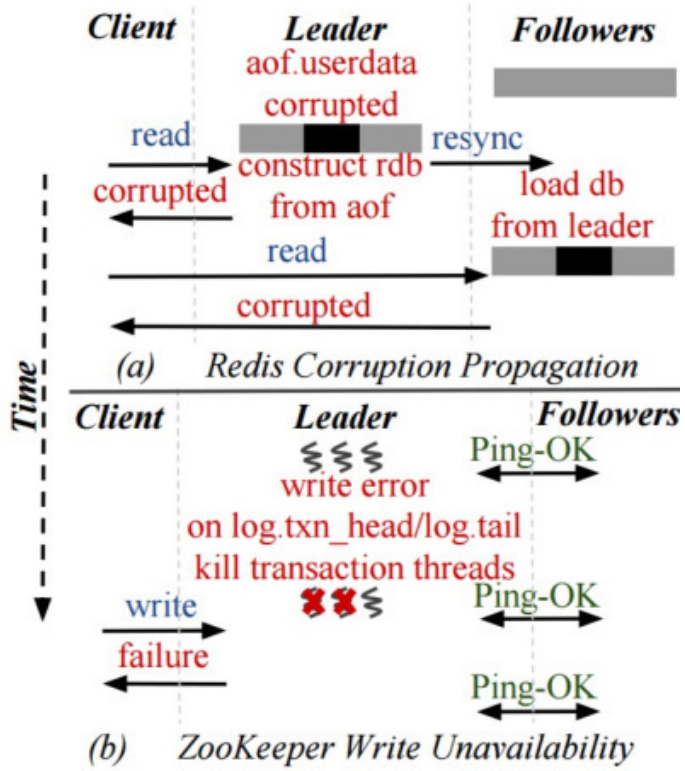


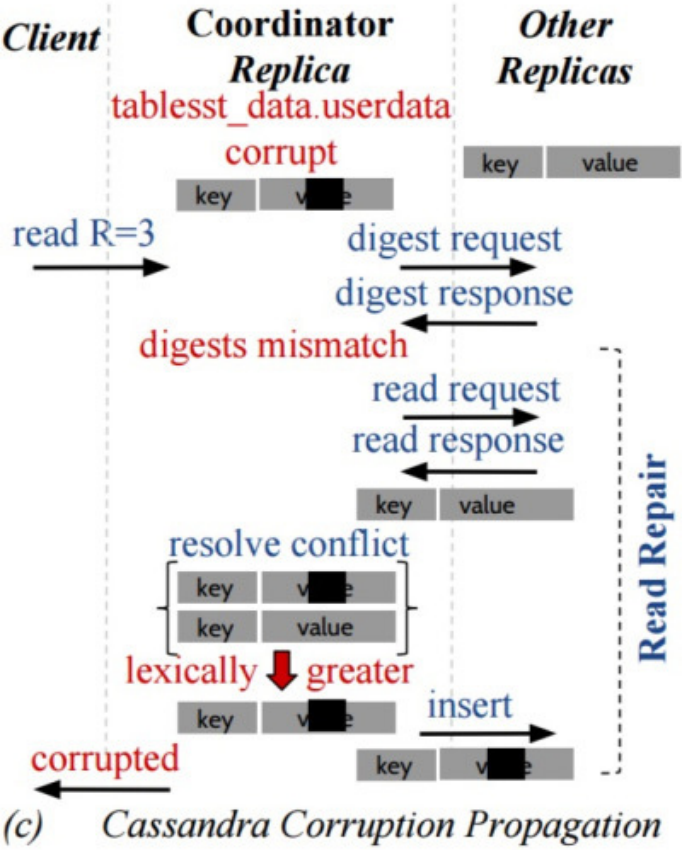
Figure 1: **System Behaviors.** *The figure shows system behaviors when corruptions (corrupted with either junk or zeros), read errors, write errors, and space errors are injected in various on-disk logical structures. The leftmost label shows the system name. Within each system workload (read and update), there are two boxes – first, local behavior of the node where the fault is injected and second, cluster-wide global effect of the injected fault. The rightmost annotation shows the on-disk logical structure in which the fault is injected. It takes the following form: file_name.logical_entity. If a file can be contained in a single file-system block, we do not show the logical entity name. Annotations on the bottom show where a particular fault is injected (L - leader/master, F - follower/slave). A gray box for a fault and a logical structure combination indicates that the fault is not applicable for that logical structure. For example, write errors are not applicable for the epoch structure in ZooKeeper as it is not written and hence shown as a gray box.*

Here are some examples of problems found.

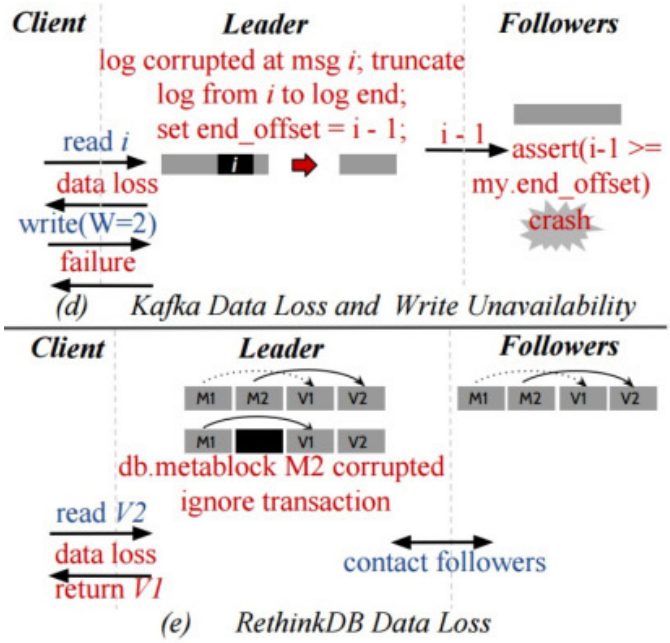
Corruption propagation in Redis, and write unavailability in ZooKeeper:



Corruption propagation in Cassandra:



Data loss in Kafka and RethinkDB



FIVE KEY OBSERVATIONS

Taking a step back from the individual system problems the authors present a series of five observations with respect to data integrity and error handling across all eight systems.

Even though the systems employ diverse data integrity strategies, all of them exhibit undesired behaviours.

Technique	Redis	ZooKeeper	Cassandra	Kafka	RethinkDB
Metadata Checksums	P	√	√	√	P
Data Checksums	P	√[a]	√[$]	√	
Background Scrubbing					
External Repair Tools	√		√		
Snapshot Redundancy	P*	P*			

P - applicable only for some on-disk structures; a - A

Sometimes, seemingly unrelated configuration settings affect data integrity. For example, in Cassandra, checksums are verified only as a side effect of enabling compression. Due to this behavior, corruptions are not detected or fixed when compression is turned off, leading to user-visible silent corruption. We also find that a few systems use inappropriate checksum algorithms. For example, ZooKeeper uses Adler32 which is suited only for error detection after decompression and can have collisions for very short strings.

On the local node, faults are often undetected, and even if detected crashing is the most common local reaction. Sometimes this leads to an immediate harmful global effect. After crashing, simple restarting may not help if the fault is sticky – nodes repeatedly crash until manual intervention fixes the underlying problem.

For instance, in Redis, corruptions in the appendonly file of the leader are undetected, leading to global silent corruption... Likewise, RethinkDB does not detect corruptions in the transaction head on the leader which leads to a global user-visible data loss.

Redundancy is under-utilised, and a single fault can have disastrous cluster-wide effects... Even if only a small amount of data becomes lost or corrupted, an inordinate amount of data can be affected.

Structures	Fault Injected	Scope Affected
Redis:		
appendonlyfile.metadata	any	All#
appendonlyfile.userdata	read, write errors	All#
Cassandra:		
tablesst_data.block_0	corruptions (junk)	First Entry$
tablesst_index	corruptions	SSTable#
schemasst_compressioninfo	corruptions, read error	Table#
schemasst_filter	corruptions, read error	Table#
schemasst_statistics.0	corruptions, read error	Table#
Kafka:		
log.header	corruptions	Entire Log$
log.other	corruptions, read error	Entire Log$*
replication_checkpoint	corruptions, read error	All$
replication_checkpoint_tmp	write errors	All#
RethinkDB:		
db.transaction_head	corruptions	Transaction$
db.metablock	corruptions	Transaction$

$- data loss # -inaccessible *- starting from corrupted entry

Table 3: **Scope Affected.** *The table shows the scope of data (third column) that becomes lost or inaccessible when only a small portion of data (first column) is faulty.*

Contrary to the widespread expectation that redundancy in distributed systems can help recover from single faults, we observe that even a single error or corruption can cause adverse cluster-wide problems...

Contrary to the widespread expectation that redundancy in distributed systems can help recover from single faults, we observe that even a single error or corruption can cause adverse cluster-wide problems such as total unavailability, silent corruption, and loss or inaccessibility of inordinate amount of data. Almost all systems in many cases do not use redundancy as a source of recovery and miss opportunities of using other intact replicas for recovering. Notice that all the bugs and undesirable behaviors that we discover in our study are due to injecting only a single fault in a single node at a time. Given that the data and functionality are replicated, ideally, none of the undesirable behaviors should manifest.

Crash and corruption handling are entangled – the detection and recovery code of many systems does not distinguish between crashes and corruptions. (They should!)

Nuances in commonly used distributed protocols can spread corruption or data loss... "*we find that subtleties in the implementation of commonly used distributed protocol such as leader-election, read repair and resynchronization can propagate corruption or data loss.*"

HOW CAN WE IMPROVE THE SITUATION?

1. "*As more deployments move to the cloud where reliable storage hardware, firmware, and software might not be the reality, storage systems need to start employing end-to-end integrity strategies.*"

2. "*...we believe that recovery code in distributed systems is not rigorously tested, contributing to undesirable behaviors. Although many systems employ checksums and other techniques, recovery code that exercises such machinery is not carefully tested. We advocate future distributed systems need to rigorously test failure recovery code using fault injection frameworks such as ours.*"

3. The body of research work on enterprise storage systems provides guidance on how to tackle partial faults, "*such wisdom has not filtered down to commodity distributed storage systems.*" (and it needs to!)

4. We're going to need a fairly fundamental redesign in some cases:

...although redundancy is effectively used to provide improved availability, it remains underutilized as a source of recovery from file-system and other partial faults. To effectively use redundancy, first, the on-disk data structures have to be carefully designed so that corrupted or inaccessible parts of data can be identified. Next, corruption recovery has to be decoupled from crash recovery to fix only the corrupted or inaccessible portions of data. Sometimes, recovering the corrupted data might be impossible if the intact replicas are not reachable. In such cases, the outcome should be defined by design rather than left as an implementation detail.

Developers of all the systems under test were contacted regarding the problems uncovered by this testing. RethinkDB are changing their design to include application level checksums. The ZooKeeper write unavailability bug was also discovered in the wild and has recently been fixed.

Related to this paper, anyone building systems that interact with a file system API should also read "All file systems are not created equal."

THE CURIOUS CASE OF THE PDF CONVERTER THAT LIKES MOZART

Harkous et al., study third-party apps that work on top of personal cloud service (e.g., Google Drive, Dropbox, Evernote,...). Careful analysis of 100 third-party apps in the Google Drive ecosystem showed that 64% of them request more permissions than they actually need.

Harkous et al., PoPET '16

No surprise there sadly! The really interesting part of the paper for me is where the authors investigate alternate permission models to discover what works best in helping users to make informed privacy choices. Their *Far-reaching insights* model (which we'll get to soon!) is a brilliant invention.

...cloud permissions allow 3rd party apps to get access to any file the user has stored in the cloud... Put simply, the scale and quality of data that can be collected is both a privacy nightmare for unaware users and a goldmine for advertisers.

The rest of this review will proceed as follows:

A quick summary of the findings from investigating existing apps and the permissions they request,

An analysis of permission models to see what best helps users to make privacy-informed choices

A brief look at PrivySeal, the privacy-informing app store the authors built for Google Drive

Suggestions from the authors for how cloud providers can improve their offerings to safeguard users' privacy.

YOU WANT ACCESS TO *WHAT*?

This study concerned Google Drive apps, but I've no reason to believe there wouldn't be similar findings in other ecosystems. Here are the possible permissions that a Google Drive app can request:

In the light of the risk that over-privileged apps pose, we propose three alternatives to the existing permission model before evaluating their efficacy.

Permission	Short Name
View and manage the **files** in your Google Drive.	DRIVE
View the **files** in your Google Drive.	DRIVE_READONLY
View and manage **metadata** of files in your Google Drive.	DRIVE_METADATA
View **metadata** for files in your Google Drive.	DRIVE_METADATA_READONLY
View and manage Google Drive files that you have **opened or created with this app**.	DRIVE_FILE
View your Google Drive apps.	DRIVE_APPS_READONLY
Add itself to Google Drive.	ADD_DRIVE
View and manage its own configuration data in your Google Drive.	DRIVE_APPDATA

Table 1. Requested permissions with the short name we use for reference

Finding out the actual permissions an app requests isn't something you can easily automate, so the authors investigated 100 randomly selected apps out of the 420 "works with Google Drive" apps in the Google Chrome Web Store at the time of the study. When we get to PrivySeal, we'll see that only around 25% of the apps that people install actually come from the official Google Web Store – and those that come from elsewhere tend to have lower privacy standards!

Analyzing the [application permissions], we found out that 64 out of 100 apps request unneeded permissions. In other words, the developers could have requested less invasive permissions with the current API provided by Google. In total, 76 out of the 100 apps requested full access to the all the files in the user's Google Drive. Moreover, the 64 over-privileged apps have actually all requested full access. Accordingly, in our sample, around 84% (64/76)of apps requesting full access are over-privileged.

FAR-REACHING INSIGHTS FOR PRIVACY-INFORMED CONSENT

As we saw yesterday, the current permissions interface of Google Drive looks like this:

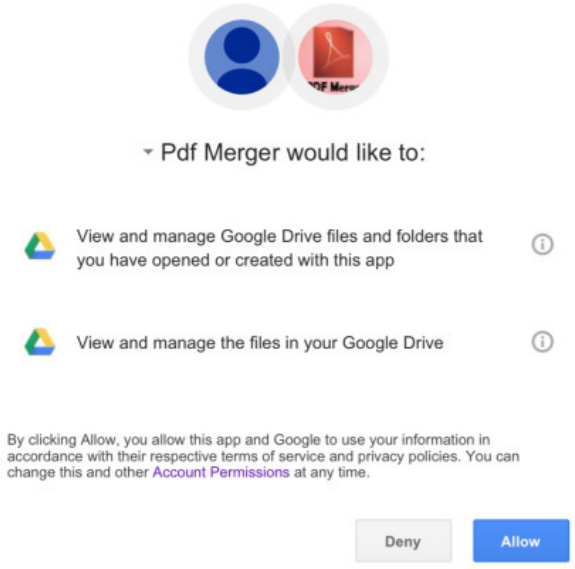


Fig. 1. Example of the current permissions interface of Google Drive

In the light of the risk that over-privileged apps pose, we propose three alternatives to the existing permission model before evaluating their efficacy.

The evaluation was done with 210 users, against a baseline of the current permissions model. The efficacy of each model is measured using an *acceptance likelihood* (AL) metric that measures the percentage of the times a user choses to actually install an app after being presented with a permissions dialog for it.

The first model is called *Delta Permissions*. In this model the permission dialog is modified to explicitly point out when an app requests more permissions than it actually needs, based on the hypothesis that users are less likely to install apps requesting unnecessary permissions.

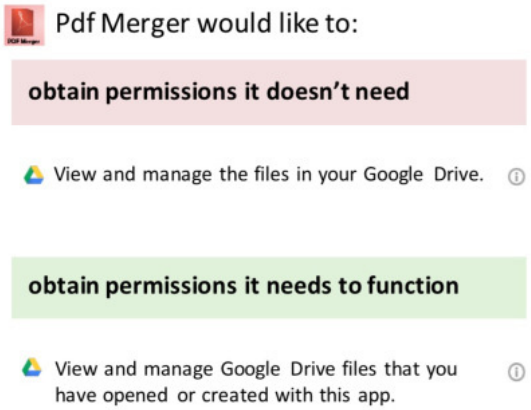


Fig. 3. Example of Delta Permissions interface

That looks like it should raise a red flag in the user's mind... And yet, "*we found no evidence of any advantage that delta permissions can introduce, which means that telling our experiment's participants explicitly about unneeded permissions did not help deter them from installing over-privileged apps.*" A great reminder of the value of doing actual studies versus just coming up with a design you think is going to work well and shipping it!

The second model is called *Immediate Insights*, based on the hypothesis that users shown samples of the data that can be extracted from the unneeded permissions are less likely to install apps requesting them. The delta permissions dialog is expanded with a new panel on the right with the question "what do the unneeded permissions say about you?" and insights that can be:

- an image selected at random from the user's image files
- a photo from the user's image files, which includes GPS location information, placed on a map
- an excerpt from the beginning of a randomly chosen text file
- the profile picture and name of a randomly chosen collaborator

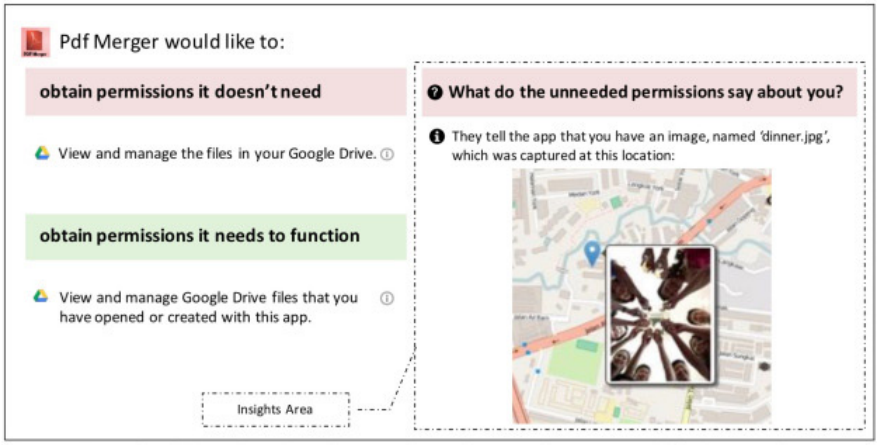


Fig. 4. Example of Immediate Insights interface; the same layout is used for Far-reaching insights, with the insights area content changing accordingly

The third model is called *Far-reaching Insights* based on the hypothesis that "*when the users are shown the far-reaching information that can be inferred from the unneeded permissions granted to apps, they are less likely to authorize these apps*." The appearance is similar to the immediate insights dialog, but the panel is replaced with one that shows deeper inferences from the data in six different categories:

Entities, concepts, and topics extracted by using NLP techniques on user's textual files:

The **Sentiment** of the user towards entities with the most positive or negative sentiments:

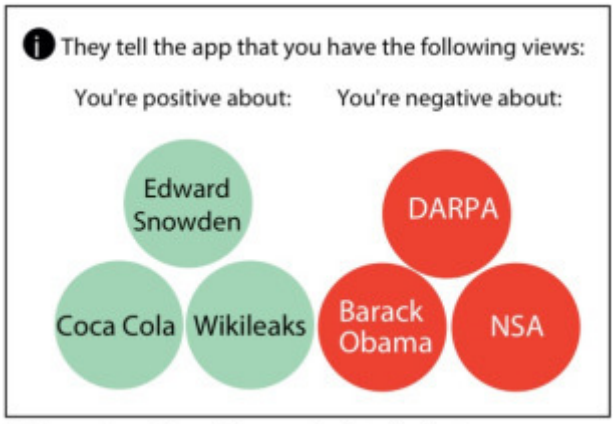


Fig. 6. *Sentiments* insight

The **top collaborators** a user has, based on the analysed files.

The user's shared interests with collaborators:

Fig. 7. *SharedInterests* insight

Faces with context – faces of the most frequent people featuring in the user's images, together with the concepts that appear in the same images.

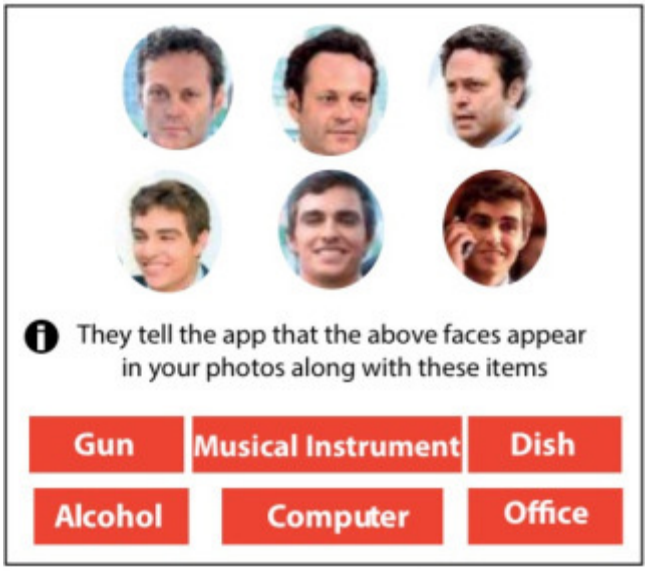


Fig. 8. *FacesWithContext* insight

Faces on a map – shows a user where selected photos were taken, together with the faces and items in those photos:

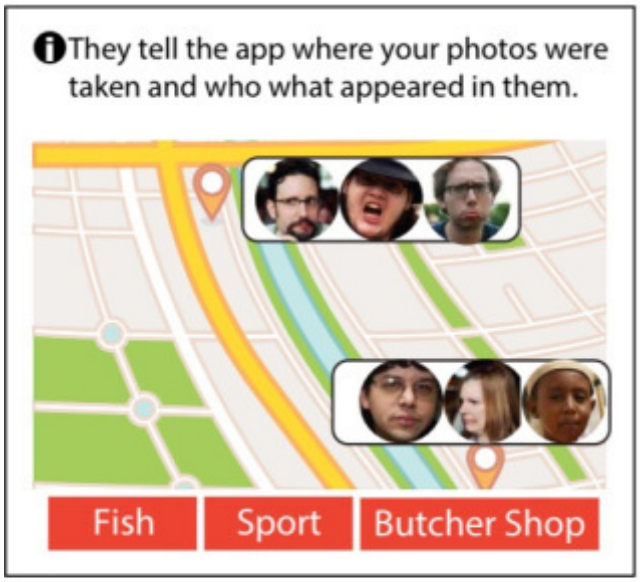


Fig. 9. *FacesOnMap* insight

The techniques used to generate the inferences are shown in the following figure.

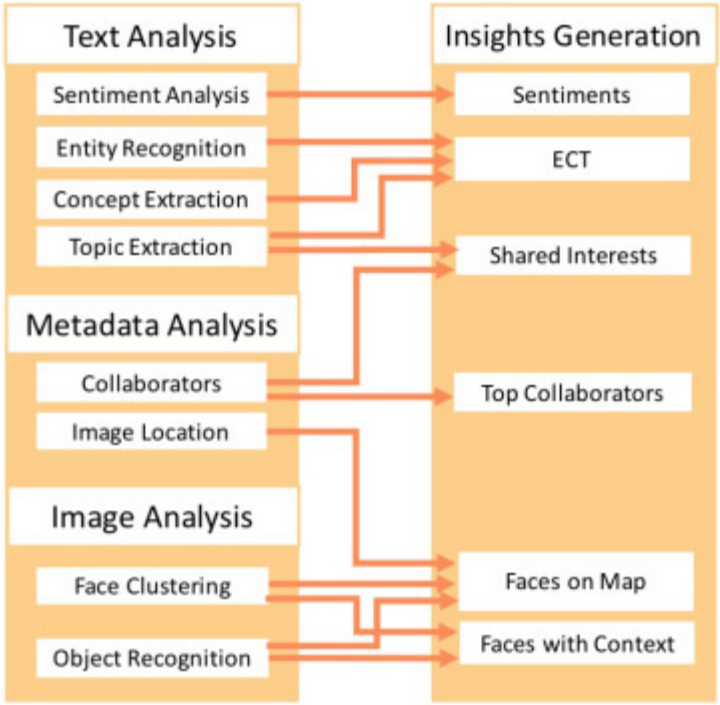


Fig. 10. Component diagram mapping the used analysis techniques to the generated insights

The experiment reveals which techniques are most successful in discouraging users from installing apps which request permissions they don't need:

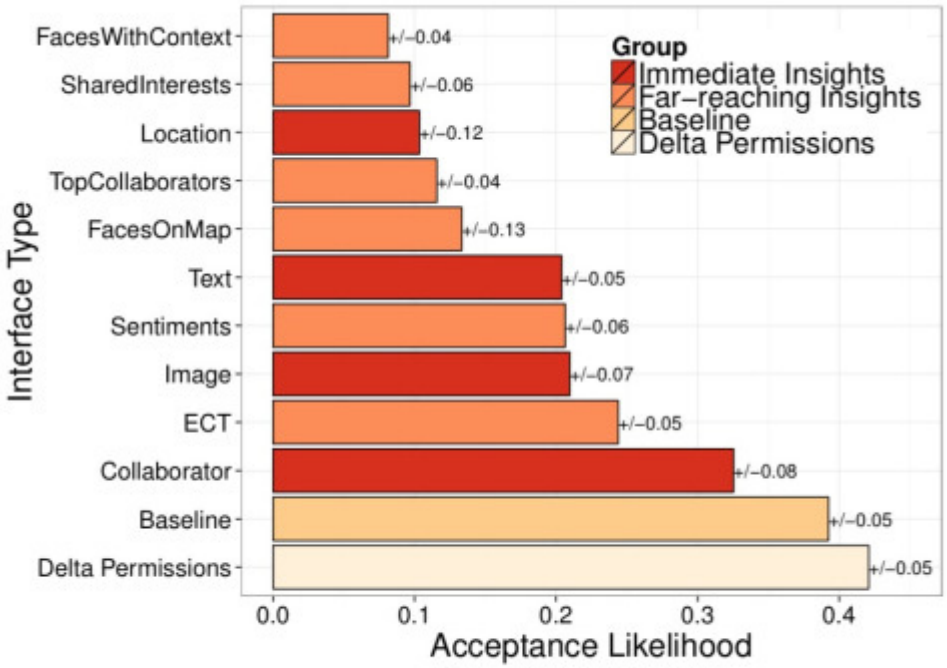


Fig. 11. *AL* for the different types of interfaces; numbers next to each bar are the error values at 95% confidence interval

Overall, immediate insights are roughly twice as good as the baseline at discouraging the acceptance of such apps, but *far-reaching insights* are up to twice as good again as the immediate insights.

The category of *personal insights* relating to the user himself or herself (image & text from immediate insights, entities-concepts-topics and sentiments from far-reaching insights) are all associated with a significantly higher acceptance likelihood than the far-reaching insights faces-with-context, top-collaborators, and shared-interests.

We denote these as relational insights. From our results, we can conclude that relational insights promote greater privacy awareness in users, as such insights are more likely to dissuade them from installing over-privileged apps.

Also interesting here is that the top-collaborators insight, shown to be something that users care about with respect to their privacy, is something that can be inferred even if an app only has *metadata* permissions, and not full file access. §7.2 of the paper explores in more detail what can be learned from metadata alone.

PRIVYSEAL

We could all wait for Google to implement something like the far-reaching insights in their app store – but that might take some time if ever (and even then, only covers a subset of apps).

... we decided not to wait and chose an alternative approach, which is independent of the company's plans and is ready for user utilization immediately. We built PrivySeal, a privacy-focused store for Google Drive apps, which is readily available at https://privyseal.epfl.ch.

To generate its far-reaching insights of course, PrivySeal *itself* requires access to all your files. If such a solution where hosted by the cloud service provider themselves, similar considerations would of course apply – unless a user-key encryption scheme is in use. From the 1440 registered users of PrivySeal at the time the paper was compiled, there were 662 unique apps installed in their Google Drive accounts (many of these installed before the users connected to PrivySeal of course). These came from the Google Chrome Web Store, other Google stores (add-ons and apps marketplace), or outside of Google's stores altogether. The following summary shows that apps in unofficial stores tend to be in the majority, as well as being much more cavalier with the permissions they request. Improving the Chrome Web Store permissions model may help a bit therefore, but an ideal solution would be independent of the various stores apps can be found in.

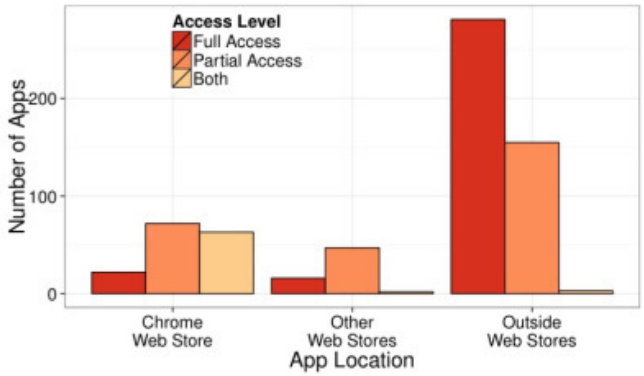


Fig. 14. Change of access level with app location

HOW CAN WE DO BETTER?

The authors recommend four steps in additional to PrivySeal that would users make informed privacy decisions:

Providing finer-grained permission models to help apps request only what they truly need

A privacy-preserving intermediate API layer that sits over the top of existing cloud service APIs providing finer grained control, permissions reviews, and *transparency dashboards*.

Transparency dashboards – a post-installation technique to deter developers from abusing user data. Transparency dashboards allow the user to see which files have been downloaded by each 3rd party app and when such operations took place. This either needs to be integrated into the platform itself, or built into a privacy preserving intermediate API that extensions work with.

Insights based on used data, "*unlike external solutions that can only determine what data can be potentially accessed, the cloud platform can provide users with insights based on the data that developers (vendors) have previously downloaded.*"

See the related work in section 9.1 of the paper for references to studies looking at user consent mechanisms and their efficacy in the context of Facebook apps, Android apps, and Chrome extensions.

The privacy informing user interfaces in this work were designed in accordance with the principles set out in "A design space for effective privacy notices," which is also well worth checking out. With new laws coming into effect soon that require *informed* consent, lets hope we start to see more of this sort of thing in the wild.

HERE IS WHAT WE'VE COVERED IN THE PREVIOUS ISSUES

DBSHERLOCK: A PERFORMANCE DIAGNOSTIC TOOL FOR TRANSACTIONAL DATABASES

GOODS: ORGANIZING GOOGLE'S DATASETS

FLEXIBLE PAXOS: QUORUM INTERSECTION REVISITED

ON DESIGNING AND DEPLOYING INTERNET SCALE SERVICES

ON DESIGNING AND DEPLOYING INTERNET SCALE SERVICES

A SURVEY OF AVAILABLE CORPORA FOR BUILDING DATA-DRIVEN DIALOGUE SYSTEMS

HOW TO BUILD STATIC CHECKING SYSTEMS USING ORDERS OF MAGNITUDE LESS CODE

DEEP LEARNING IN NEURAL NETWORKS: AN OVERVIEW

THE AMAZING POWER OF WORD VECTORS

GORILLA: A FAST, SCALABLE, IN-MEMORY TIME-SERIES DATABASE