Meerkat:
Detecting Website Defacements through Image-based Object Recognition

Kevin Borgolte, Christopher Kruegel, Giovanni Vigna
University of California, Santa Barbara
{kevinbo,chris,vigna}@cs.ucsb.edu

Abstract

Website defacements and website vandalism can inflict significant harm on the website owner through the loss of sales, the loss in reputation, or because of legal ramifications.

Prior work on website defacements detection focused on detecting unauthorized changes to the web server, e.g., via host-based intrusion detection systems or file-based integrity checks. However, most prior approaches lack the capabilities to detect the most prevailing defacement techniques used today: code and/or data injection attacks, and DNS hijacking. This is because these attacks do not actually modify the code or configuration of the website, but instead they introduce new content or redirect the user to a different website.

In this paper, we approach the problem of defacement detection from a different angle: we use computer vision techniques to recognize if a website was defaced, similarly to how a human analyst decides if a website was defaced when viewing it in a web browser. We introduce MEERKAT, a defacement detection system that requires no prior knowledge about the website's content or its structure, but only its URL. Upon detection of a defacement, the system notifies the website operator that his website is defaced, who can then take appropriate action. To detect defacements, MEERKAT automatically learns high-level features from screenshots of defaced websites by combining recent advances in machine learning, like stacked autoencoders and deep neural networks, with techniques from computer vision. These features are then used to create models that allow for the detection of newly-defaced websites.

We show the practicality of MEERKAT on the largest website defacement dataset to date, comprising of 10,053,772 defacements observed between January 1998 and May 2014, and 2,554,905 legitimate websites. Overall, MEERKAT achieves true positive rates between 97.422% and 98.816%, false positive rates between 0.547% and 1.528%, and Bayesian detection rates[1] between 98.583% and 99.845%, thus significantly outperforming existing approaches.

1 Introduction

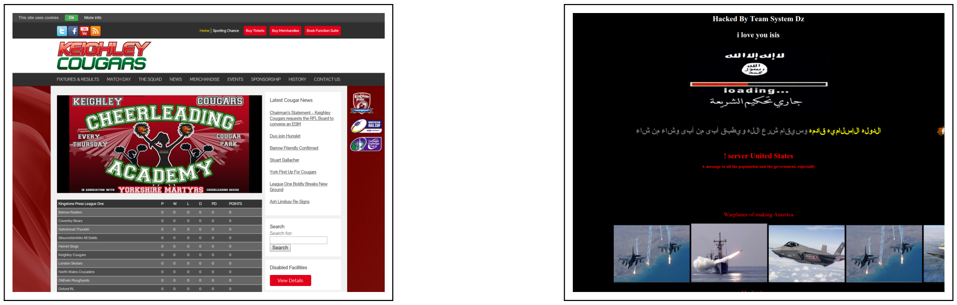
The defacement and vandalism of websites is an attack that disrupts the operation of companies and organizations, tarnishes their brand, and plagues websites of all sizes, from those of large corporations to the websites of single individuals [1–3].

In a website defacement, an attacker replaces the content of a legitimate website with some of his/her own content. A website might be defaced for many different reasons and in many different ways: For example, an attacker might deface the website by brute-forcing the administrator's credentials, by leveraging a SQL injection to introduce content or code, or by hijacking the domain name; however, all defaced websites share one characteristic: the defacer leaves a message that is shown to the visitors of the website instead of the legitimate content, changing the visual appearance of the website.

Although nearly all defacers vandalize websites for their "15 minutes of fame," and to get a platform to publicize their message, their messages vary: some embarrass the website's operator, others make a political or religious point, and others again do it simply for "bragging rights." For instance, in the beginning of November 2014, as reported by the BBC [4], attackers defaced the website of the Keighley Cougars, a professional rugby club from England competing in League 1. The defacers modified the website so that visitors were greeted with a message in support of the terrorist organization "Islamic State of Iraq and the Levant/Syria" (ISIL/ISIS). Similarly, in late 2012, defacers close to the Syrian regime defaced the homepage of the prominent Qatari television network Al Jazeera, and instead of being shown news articles, visitors were greeted by a message alleging Al Jazeera of "spreading false fabricated news." Reliably detecting such website defacements is challenging, as there are many ways in which an attacker can tamper with the website's appearance, including re-routing the traffic to a different website, which does not affect the legitimate website's content directly in any way.

In this paper, we introduce MEERKAT, a website monitoring system that automatically detects if a website has been defaced. MEERKAT detects website defacements by rendering the website in a browser, like a normal visitor

[1] The Bayesian detection rate is the likelihood that if we detect a website as defaced, it actually is defaced, i.e., P(true positive|positive).


KEIGHLEY COUGARS
CHEERLEADING ACADEMY
Hacked By Team System Dz
i love you isis
loading...
جاري تحكيم الشريعة
الدوله الإسلاميه قادمه وسيقام شرع الله ويطبق أبى من أبى وشاء من شاء
! server United States
A message to all the population and the government, especially
Warplanes of making America

(a) Normal, non-defaced version.

(b) Defaced version.

Figure 1: Screenshots of the Keighley Cougars homepage, non-defaced and defaced in an attack on November 2, 2014. (a) shows how the website looks normally, (b) shows how the defaced website looked like after being defaced by *Team System Dz*, a defacer group close to the terrorist organization Islamic State of Iraq and the Levant/Syria (ISIL/ISIS).

would, and deciding, based on features learned exclusively from screenshots of defacements and legitimate websites observed in the past, if the website's *look and feel* is that of a defaced or a legitimate website. If the website is detected as being defaced, the system notifies the operator, who, in turn, can, depending on the confidence in MEERKAT's decision, put the website (automatically) in maintenance mode or restore a known good state to reduce the damage.

Our technical contributions in this paper are:

- We introduce MEERKAT, a website defacement detection system that learns a high-level feature set from the visual representation of the website, i.e., it learns a compressed representation of the *look and feel* of website defacements and legitimate websites. Based on the learned features, the system then produces a model to differentiate between defaced and legitimate websites, which it uses to detect website defacements in the wild. In addition, the system notifies the website's operator upon detection (Section 3).

- We evaluate MEERKAT on the largest website defacement dataset to date, comprising of 10,053,772 website defacements observed between January 1998 to May 2014, and 2,554,905 legitimate and (supposedly) not defaced websites from Alexa's, MajesticSEO's, and QuantCast's top 1 million lists (Section 4).

In the remainder of this paper, we make a compelling case for the need of an accurate and lightweight website monitoring system that detects website defacements (Section 2), discuss how MEERKAT works in detail (Section 3), evaluate our system on the largest defacement dataset to date (Section 4), discuss some limitations of website defacement detection systems (Section 5), compare MEERKAT to related work (Section 6), and, finally, we conclude (Section 7).

2 Motivation

Lately, the detection of website defacements as a research topic has not received much attention from the scientific community, while, at the same time, defacements became more prominent than they have ever been. The number of reported defacements has been exceeding the number of reported phishing pages since October 2006 by a factor of 7 on average, and reached up to 33.39 defacements being reported to Zone-H[2] per phishing page reported to PhishTank[3] (see Figure 2). Yet, website vandalism is often played down as a problem instead of being acknowledged and addressed.

The increase in defacements is evident (see Figure 2): while a mere 783 verified defacements were reported on average each day to Zone-H in 2003, the number of reports increased to 3,258 verified defacements per day for the year 2012, to an all-time high of over 4,785 verified defacements being reported each day to Zone-H in 2014. This corresponds to an increase of websites being defaced by 46.87% from 2012 to 2014 [5].

Similarly, according to the Malaysian Computer Emergency Response Team (CERT), 26.04% of all reported incidents in 2013 were website defacements, but only 1.5% of the reported incidents were defacements in 2003, and 10.81% were website defacements in 2007 [7, 8].

Furthermore, in 2014, attackers confirmedly defaced over 53,000 websites ranked on Alexa's, MajesticSEO's, and QuantCast's top 1 million lists. Corroborating that not only websites that are "low-hanging fruit" are being defaced, but that high-profile ones are being attacked alike (see Table 1).

This recent resurgence and the increase in defacements and "cyber-vandalism" is generally attributed to the rise of hacktivist groups, like *anonymous* or *LulzSec* [9, 10], but also gained traction through the escalation of international conflicts [11, 12]. Although the scientific consensus is that the attacks employed to deface a website are usually rather primitive in nature [9], hacktivist groups and other politically- and religiously-motivated defacers have been extremely successful in the past: in February 2015, Google Vietnam was defaced by *Lizard Squad* for several hours [13]; in January 2015, the website of Malaysia Airlines was defaced by *Cyber Caliphate* [3]; in late 2014, the defacer group *Team System Dz* defaced over 1,700 websites to speak out against the actions of the US in the Syrian civil war and to advocate for ISIS/ISIL [2]; in April 2014, over 100 websites be-

[2] Zone-H [5] is an archive containing only defaced websites, all reported defacements are mirrored locally and manually verified [6]. Upon manual inspection, a reported defacement is removed from the archive if it does not constitute a defacement, or it is marked as verified.

[3] PhishTank is the largest public clearinghouse of data about phishing scams, users report potential phishing scams and other users agree or disagree with the submitter, resulting in a user-assigned phishing score. Phishing pages are not being verified by expert analysts.

Month	Website	Alexa		MajesticSEO		QuantCast	Page Views per Month ▼
		US	Global	TLD[1]	Global	US	
Nov 2014	`princeton.edu`	999	3,412	17	273	3,444	796,000
	`volvo.com`	54,607	57,046	3,757	7,323	568,058	-
	`cca.gov.in`[2]	146,039[3]	780,660	-	-	-	-
Aug 2014	`openelec.tv`	7,226[4]	48,754	184	93,894	-	-
	`omicsonline.org`	7,561[3]	42,030	5,068	63,924	-	-
Jul 2014	`ct.gov`	2,454	10,976	72	2,054	3,548	809,000
	`us.to`	2,846[5]	28,100	18	11,061	-	-
	`sunnewsonline.com`	68[6]	9,958	31,315	58,277	236,740	-
	`newsmoments.in`	3,725	39,262	-	-	-	-
Jun 2014	`wordpress.net`	3,522[7]	41,295	1,410	28,021	321,317	-
May 2014	`arynews.tv`	72[8]	5,308	949	536,436	-	-
	`sundaytimes.lk`	120[9]	38,591	6	39,866	209,083	-
Mar 2014	`taylorswift.com`	3,560	23,425	12,161	23,608	15,678	1.2 million
	`gbjobs.com`	798[10]	9,181	-	-	-	-
Dec 2013	`openssl.net`	5,994	16,409	80	933	-	-
Oct 2013	`avg.com`	117	155	471	854	-	37 million
	`aljazeera.net`	25[11]	1,831	37	920	2,196	28 million
	`bitdefender.com`	5,934	5,898	1,132	2,094	3,963	1.4 million
	`avira.com`	24[12]	1,108	1,275	2,361	6,081	480,000
	`leaseweb.com`	359[4]	4,035	23,585	44,451	230,626	-
	`metasploit.com`	124,365	175,570	33,537	59,816	120,839	-
2011-2013[14]	`telegraph.co.uk`	21[13]	225	3	107	6[13]	125 million
	`ups.com`	71	231	319	549	101	40 million
	`nationalgeographic.com`	483	1,006	94	139	125	37 million
	`acer.com`	4,060	6,042	-	-	1,995	2.9 million
	`theregister.co.uk`	2,737	3,457	443	14	11,327	1 million
	`vodafone.com`	7,052[13]	20,625	5,833	2,980	101,624	-

Table 1: Recent high-profile websites that were defaced, with their respective page rank according to Alexa, MajesticSEO, and QuantCast, and their monthly page impressions. These defacements were reported to Zone-H and include a major logistics company (UPS), computer and information security vendors (BitDefender, Avira, AVG, MetaSploit), news websites (Al Jazeera, Ary News, News Moments, Sunday Times, Sun News Online, Telegraph, The Register), a scientific society (National Geographic), a hardware vendor (Acer), the world's second largest telecommunications provider (Vodafone), a singer-songwriter/actress (Taylor Swift), the state of Connecticut (ct.gov), an Indian federal ministry (cca.gov.in), an auto-mobile company (Volvo), an ivy-league university (Princeton), well-known open source projects (OpenSSL, OpenELEC), and a hosting provider (Leaseweb). Missing fields represent unavailable data, data is unavailable due to being kept secret by the website operators or requiring subscriptions to Alexa, MajesticSEO or QuantCast.

[1] Top-level domain rank. [2] Government of India, Ministry of Communications & Information Technology. [3] Rank in India. [4] Rank in Netherlands. [5] Rank in Indonesia. [6] Rank in Nigeria. [7] Rank in Bulgaria. [8] Rank in Pakistan. [9] Rank in Sri Lanka. [10] Rank in China. [11] Rank in Yemen. [12] Rank in Iran. [13] Rank in United Kingdom. [14] Selected high-profile website defacements from Fortune 50 and Global 500 companies between 2011 to 2013.

longing to the government and major companies in Zambia were defaced by Syrian and Saudi Arabian defacers to voice against the Western world's *meddling* in the Syrian civil war [14]; in January 2014, the website of the popular mobile game Angry Birds was defaced in protest of governmental spying by the NSA and GHCQ [15]; and, in October 2013, a Pakistani defacer group gained access to the domain registrars of Suriname, Antigua & Barbados, and Saint Lucia and defaced the regional websites of Audi, AVG, BlackBerry, BMW, Canon, Coca-Cola, Fujitsu, Hitachi, Honda, IBM, Intel, Microsoft, Samsung, Symantec, Rolls-Royce, Vodafone, and other companies simply for "bragging rights" [16].

A prime example that quantifies the impact of defacements is the case of the Telegraph, a major UK daily newspaper, which was defaced in September 2011. The Telegraph is the third most-visited website in the United Kingdom, according to MajesticSEO, and it is the 21st most visited website in the United States, according to Alexa. Each month, the homepage of the Telegraph is visited over 125 million times (48 times per second), and, since reports state that the defacement lasted around three hours, we can estimate that more than 500,000 people saw the defacement instead of the legitimate website.[4]

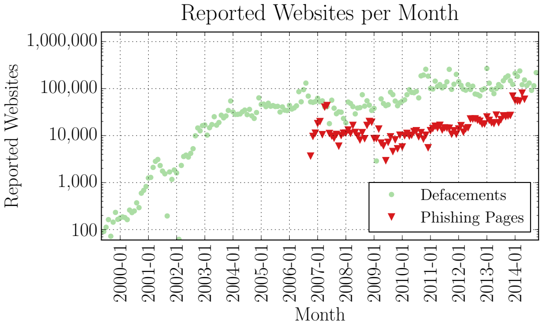


Figure 2: Defacements reported to Zone-H and phishing pages reported to PhishTank, per month from January 2000 to including October 2014. The drops in reported defacements in February 2002, February 2009, and March 2009 are because Zone-H was under maintenance during that time and did not accept any new reports. No data is available from PhishTank earlier than October 2006, when the website was launched. The trend of an increasing number of defacements per month, as well as the gap in the number of defacements to the number of phishing pages of a factor of up to 33x are evident.

While the list of prominent defacements goes on [4, 17–25], it is important to note that most techniques to deface a website, like code and data injection attacks (such as SQL injections), improper access control, or DNS hijacking and poisoning, have been well-studied and protection mechanisms have been proposed by prior

[4] Since the website was defaced on a Sunday afternoon local time in the United Kingdom, the number of visitors is likely much higher.

work [26, 27]. However, it is extremely hard to protect against all defacement attacks simultaneously and at scale.

Even worse, organizations are often responsible for hundreds (or thousands) of different websites, with different levels of security [9]. A single insecure website that is defaced, however, can inflict significant harm on the organization: in qualitative terms because of the loss of reputation, and in quantitative terms because of the cost of having to investigate and remove the defacement.

Although defacements can inflict serious harm on the website operator, a two-month study by Bartoli et al. [28] shows that many website operators still react slowly to defacements with an average response time of over 72 hours. Moreover, their study finds that mere 24% of the defaced websites were restored within one day, about 50% defacements were removed within the first week, while more than 37% of the websites remained defaced for over two weeks. Overall, their findings suggest that prior website defacement protection techniques and detection methods have not been widely adopted.

We argue that the logical first step is to reduce the harm inflicted on the website operator by quickly detecting if his/her website has been defaced, so that the operator can put the website in maintenance mode or restore its content to a known good state. As such, an automatic, accurate, and lightweight defacement detection system that monitors websites, notifies the website's operator, and acts as an early warning system is desired. In this paper, we propose one such system, MEERKAT.

3 Meerkat

The approach MEERKAT takes to detect website defacements is fundamentally different from prior work for three reasons. First, while the system does leverage machine learning for classification, it does not rely on handpicked features that were selected based on prior domain knowledge, i.e., it requires no feature engineering. Instead, MEERKAT relies on recent advances in machine learning, stacked autoencoders, to learn high-level features directly from data. Second, MEERKAT does not require the website operator to supply any information other than the domain name at which his/her website can be accessed. We designed our system in this way because other defacement detection systems that require the operator to define keywords and other metadata, provide a reference version of his/her website, or describe the website's legitimate content, have been rarely adopted in the past. By reducing the effort required from the website operator to actually use a defacement detection system, we hope to improve on this situation. Finally, MEERKAT approaches defacement detection visually: the system analyzes the *look and feel* of the website and how a user would experience it by rendering it in a web browser and analyzing a screenshot of the website, instead of analyzing its source code or content.

Approaching the problem of detecting website defacements visually has several advantages over analyzing the source code or content of a website: some defacements rely heavily on JavaScript and Cascading Style Sheets (CSS) to stylize the defacement, which all must be analyzed in an overarching browser context, and others again rely heavily on images. In fact, similar to spam, phishing, and many scams, defacements often do not contain much textual content, but include images to display text instead [29], thus they trivially evade text-based detection approaches. Furthermore, the source code of two websites can be vastly different, yet they appear the same to the human eye when rendered in a browser. Therefore, leveraging prior work, such as DELTA [30], to analyze the DOM-tree, the website's code, or parts thereof, is unlikely to be successful when trying to detect website defacements accurately, which is why we opted for a perceptual approach that does not suffer from the aforementioned problems.

Following, we describe how MEERKAT learns from defacements and legitimate websites, and how it detects defacements in the wild. Next, we motivate the structure of our deep neural network briefly, then, we discuss the concept and motivation of fine-tuning the network, then, we provide some notes on our implementation, and, last, we briefly recap how MEERKAT can be deployed in practice.

3.1 Training and Detection

Before MEERKAT can be trained, two crucial parameters must be selected that determine how and from what data the system learns the *look and feel* of defacements:

Window Size. MEERKAT is not trained on whole screenshots of websites, but on a window "into" each website (i.e., only a part of the screenshot), thus we must select the size of these *representative windows*. Some important considerations must be made before picking the size of the windows that we extract.

A small window can be more accurate because it might only contain the exact representative part of the defacement but not any noise, like an irrelevant background color. However, if the windows are too small, the system will also have more false positives because the windows are not representative of defacements; instead, they are representative for only parts of the defacements, which might also occur in legitimate websites.

On the other hand, when using larger windows, it will take significantly longer to train the network initially, but the network might learn a more accurate model. However, if the windows are too large, then the system will learn about specific kinds of defacements in-detail and overfit; e.g., the system might learn that two defacements are different, while the two defacements are actually the same but have a slightly different, dynamically-generated background image.

Considering the trade-offs for different window sizes, for our implementation, we decided to extract windows that are `160×160` pixels in size. Our evaluation later shows that this window size works well in practice to detect website defacements (see Section 4). We briefly explored other window sizes, like `30×30`, that fared worse.

Window Extraction Strategy. The strategy to extract the representative window from a screenshot is fundamental to learn the *look and feel* of defacements and legitimate websites. If the windows are extracted according to some poorly chosen strategy, then we expect the classification accuracy to be poor as well. For instance, if the strategy always extracts the part of a website that is just a plain background, the system will only detect plain backgrounds. Therefore, it is crucial that the window extraction strategy is chosen well, and we compare some suitable strategies, like extracting the window always from the center or at random, later (see Section 3.1.2).

After selecting these parameters carefully, the system can be trained. This is where most of the complexity of MEERKAT lies. The training phase works as follows:

1. We collect a considerable amount of labeled website defacements and legitimate websites, and we extract their graphic representation (i.e., a screenshot of the browser window; Section 3.1.1)

2. For each sample, we extract the `160×160` representative window from each screenshot according to the selected extraction strategy (Section 3.1.2).

3. The representative windows are first used to learn the features of our approach, and then to learn the model for classification, for which we use a neural network (Section 3.2).

Once the neural network is trained, MEERKAT detects defacements in the wild. Its detection phase consists of only two steps, on which we expand later:

1. The website is visited with a browser to retrieve a representative screenshot (Section 3.1.1).

2. A sliding window approach is used to check if the website is defaced and, if so, an alert is raised (Section 3.1.3).

3.1.1 Screenshot Collection

The first step to detect if a website has been defaced based on its *look and feel* is to collect a screenshot of how the website looks for a normal visitor. MEERKAT visits the website with a browser that renders the website like any other browser would, and takes a screenshot once the browser finished rendering the website. In our implementation, we use PhantomJS to collect the screenshots of the websites. PhantomJS is a headless browser based on the Webkit layout engine that renders websites (nearly) identical to Safari or Google Chrome. PhantomJS also executes included JavaScript code, renders Cascading Style Sheets (CSS), and includes dynamic content, such as advertisements, like a browser that a human would use.

Another important aspect in collecting a representative screenshot of a website with a headless browser is the resolution of the *simulated* screen. The resolution of the display is important when collecting screenshots because many websites render differently for different screen sizes, such as for mobile devices, tablets, small notebooks, or large displays. In our case, we decided to fix the resolution to `1600×900` pixels, which is a display resolution often found in budget and mid-range notebooks.

3.1.2 Window Extraction Techniques

For training the system, after collecting the screenshots, we need to extract a representative window from each screenshot so that we can train the neural network to detect defacements. Various techniques can be used to extract the representative window, which can be grouped into deterministic and non-deterministic techniques. Hereinafter, we discuss the trade-offs for four possible techniques: (i) selecting the center window, (ii) selecting n non-overlapping windows according to some measure (explained later), (iii) uniformly selecting the window at random, and (iv) randomly sampling the window's center from a Gaussian distribution for the x and y dimension separately.

Deterministic Window Extraction

The most straightforward deterministic technique is to always extracts the window from the center of the screenshot of the website. However, this makes evading the system trivial. Generally, if an attacker can accurately predict the window that will be extracted, he can force the system to learn about defacements poorly, and, in turn, deteriorate classification performance drastically. Therefore, such a simple technique is unsuitable for a detection system in an adversarial context.

Alternatively, one can extract the window according to some measure. Identifying the most representative window according to a measure (e.g., the Shannon entropy), however, forces us to compute it for all possible windows and then pick the top ranking one. In turn, for a `1600×900` screenshot and a `160×160` window, we would need to evaluate over 1 million candidate windows for each sample in the dataset. In total, for our dataset, this would require over 13 *trillion* computations of the measure just to extract the representative windows. Clearly, this is impractical.

Nonetheless, a deterministic selection strategy based on a clever measure can increase the accuracy of the system, and it can also be extended trivially to extract multiple top-ranking windows at no additional cost. However, using more than one window per sample increases the dataset size by a factor of n and prolongs training time. Therefore, n would have to be chosen carefully.

Taking into account the trade-offs the different deterministic extraction strategies bear (increased training/detection time, ease of evasion, or computationally impractical) and considering that a comprehensive evaluation of them would require at least an order of magnitude of additional experiments,[5] we decided to select a non-deterministic extraction strategy that follows intuition and is based on user interface and user experience design principles instead. This selection makes our classification performance a lower bound: other window extraction strategies might be more accurate and/or robust, but (at the same time) they also incur significant additional cost at training and/or detection time.

[5] Performing these additional experiments would require at least 6 months just in computational time on our current GPU infrastructure, which is why we decided against performing them.

Non-deterministic Window Extraction

A straightforward non-deterministic strategy to extract a window from a screenshot is to select it uniformly at random. However, one cannot simply take any point from the website's screenshot as the center of the window. Instead, it must be sampled so that the whole window contains only valid data, forcing us to sample its center from the interval $[80, 1520]$ for x and $[80, 820]$ for y (these intervals are specific to the screenshot (`1600×900`) and window size (`160×160`)). Therefore, pixels at the border have a slightly lower probability to occur in a window than those in the center. Although this is an unintended side effect, it has negligible impact in practice because the center of a website is more likely to be descriptive anyways. Alternatively, we could create an "infinite" image by wrapping the screenshot at its borders, which would, however, yield artifacts because we would combine parts of the top of the website with parts of the bottom (and left and right, respectively), resulting in windows that do not occur on the real website, which, in turn, might disturb or confuse detection.

Alternatively to selecting the window's center uniformly at random, one can sample it from any other distribution, discretizing the sampled point. For instance, from a Gaussian distribution to extract windows from mostly the center of the screenshot, but not extracting from it exclusively. A focus on the center of the website is often desirable because it is likely to be more descriptive of the website's *look and feel*. For robustness, however, we also want to the system to not learn exclusively from the center but to also learn about defacements that occur at the border of the website. Therefore, for our implementation, we extract a single window per website with a Gaussian extraction strategy with $\mu_x = 800$ and $\sigma_x = 134.63975$ for x and $\mu_y = 450$ and $\sigma_y = 61.00864$ for y, so that the windows at the border of the screenshot have a lower probability to be sampled but are not ignored completely. If x and y values outside of the screenshot are sampled, we simply resample the value for x or y respectively. We selected the μ and σ values this specifically so that we sample values outside of the screenshot only with likelihood 0.0001%.

3.1.3 Defacement Detection

After MEERKAT has been trained on a set of extracted windows, it can detect if a website has been defaced. Detecting website defacements with MEERKAT is conceptually extremely simple:

1. We visit the website that we want to check with our browser and we take a screenshot of the rendered website (Section 3.1.1).

2. We apply a standard sliding window detection approach on the screenshot we took to check if a part of the screenshot is detected as being defaced, similarly to prior work in image classification [31].

3. If a window is detected to be a defacement by MEERKAT, we raise an alert and inform the website operator that his/her website has been defaced.

Note that MEERKAT does not compare a possibly-defaced website to an older, legitimate version of it, and, thus, does not need to analyze or store an older version. Instead, it detects defacements solely by examining how the current version looks like.

Exclusively to improve performance, instead of starting in a corner of the screenshot, our system starts in the center and moves outward. This behavior is motivated by the fact that the center of the website is likely more descriptive, and our training set was focused on the center region of the screenshots. This does not mean, however, that MEERKAT misses defacements that are at the border of a website, they will be detected when the sliding window reaches the actually-defaced part, the border. The same is also true if a website is only partially defaced: once the sliding window reaches the defaced area, MEERKAT detects that the website is defaced.

Additionally, a special case worth mentioning is that a legitimate website might show a large promotional screen or an advertisement with the same intention of a website defacer: attracting attention. In turn, such a promotional screen might be similar in its *look and feel* to that of a website defacement. While MEERKAT might currently (theoretically) mislabel them as defacements, our evaluation shows that they do not matter much (see Section 4). Furthermore, if they start to matter at one point in the future, it is straightforward to consider them: the defacement engine can make use of an advertisement blocker, and the website operator could whitelist the system to not be shown any promotional screens.

3.2 Neural Network Structure

In this section, we briefly discuss the design of our deep neural network and how the different layers of the network interact with the input image. The structure of our deep neural network was notably inspired by prior work by Le at al. [32], Krizhevsky et al. [33], Sermanet et al. [31], and Girshick et al. [34]. We refer to them for further details.

The main components of our deep neural network are autoencoders, which we stack on top of each other, and a standard feed-forward neural network. Autoencoders are a special type of neural network that are used for unsupervised learning. The goal of an autoencoder is to find a compressed, possibly approximated encoding/representation of the input, which can be used to remove noise from the input, or, when autoencoders are stacked, they can learn high-level features directly from the input, like where edges in an image are, or if cats or human faces are part of an image [32].

Overall, the structure of our deep neural network is based on the following idea: first, we use a stacked autoencoder to denoise the input image and learn a compressed representation of both defaced and legitimate websites, i.e., we leverage the stacked autoencoder to learn high-level features, similar to Le et al. [32]; second, we utilize a feed-forward neural network with dropout for classification, similar to Krizhevsky et al. [33].

The initial layer of our stacked autoencoder is comprised of local receptive fields. This layer is motivated by the need to scale the autoencoders to large images [32, 35–38], this layer groups parts of the image to connect to the next layer

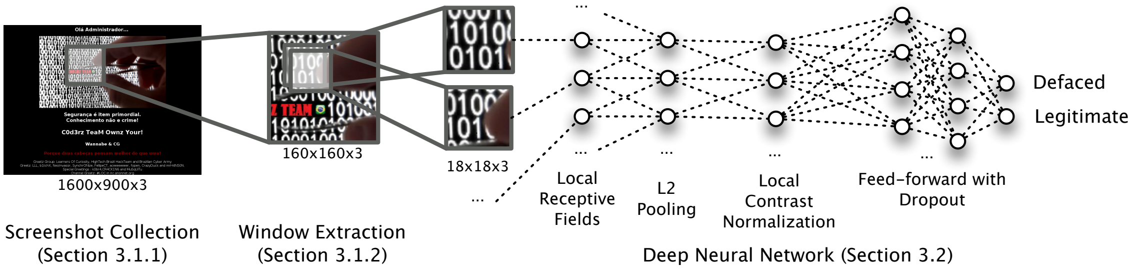

1600x900x3
160x160x3
18x18x3
...
Local Receptive Fields
L2 Pooling
Local Contrast Normalization
Feed–forward with Dropout
Defaced
Legitimate
Screenshot Collection (Section 3.1.1)
Window Extraction (Section 3.1.2)
Deep Neural Network (Section 3.2)

Figure 3: Architecture of our deep neural network.

of the autoencoder, instead of allowing the whole image to be used as input to each node of the following layer. It takes 20,164 (142^2) sub-images of size 18×18 as input, extracted at a stride of 1 from the 160×160 representative window (see Figure 3; note that each pixel in each sub-image has three dimensions for the three colors: red, green, and blue). The second layer of our stacked autoencoder employs L2 pooling to denoise local deformations of the image and to learn invariant features [37, 39–41]. Finally, the last layer of our autoencoder performs local contrast normalization for robustness [42].

The output of the stacked autoencoder is then used as the input to a feed-forward neural network with dropout that provides a 2-way softmax output. The 2-way softmax output corresponds to the two classes that we want to detect: defaced websites and legitimate websites. We use dropout in our deep neural network to prevent overfitting of the network, and to force it to learn more robust features by preventing neurons to rely that other neurons of the network are available (i.e., to prevent the co-adaptation of neurons) [43].

3.3 Fine-Tuning the Network's Parameters

In an adversarial context, such as when trying to detect if an attacker defaced a website, concept drift can be introduced intentionally by the attacker and impede the accuracy of the detection system drastically. Furthermore, concept drift also occurs naturally, such as when the style of defacements evolves over time in such a way that the features cannot distinguish between legitimate and defacement anymore. Therefore, concept drift can be a severe limitation of any detection system, if it is not taken into account and addressed properly (see Section 5.1).

MEERKAT can deal with concept drift in two different, fully-automatic ways: fine-tuning the network's parameters (adjusting feature weights), and retraining the entire network on new data. While the latter is conceptually straightforward and addresses all kinds of concept drift, it is computationally very expensive. The former, on the other hand, allows us to deal with some forms of concept drift gracefully and is computationally much less expensive. However, it requires some further attention: when fine-tuning the neural network, MEERKAT does not learn new features, but adjusts how important the already learned features are. Therefore, fine-tuning cannot address major concept drift for which the already learned features do not model defacements accurately anymore. Instead, when we fine-tune the network's parameters, we adjust the already learned weights of the deeper layers of the neural network so that new observations of defacements and legitimate websites are classified properly. As such, fine-tuning the network to maintain an accurate detection performance requires no additional information about the websites at all, but only defacements and legitimate websites that were not part of the training set before.

Conceptually speaking, when fine-tuning the network given new defacements and legitimate websites, we search for a better and, given the new data, *more optimal* set of weights in the space of all possible weights. To do so more efficiently, instead of initializing the weights at random, we initialize them based on the previously-learned weights.

3.4 Implementation

For this paper, we implemented a prototype of MEERKAT using Python and the "Convolutional Architecture for Fast Feature Embedding" (Caffe) framework by Jia et al. [44]. Caffe was used because of its high-performance and ease of use, however, it does not offer all functionality that our neural network requires and some modifications were made.

Overall, the general architecture of MEERKAT is embarrassingly parallel: the screenshot collection engine is completely separate from the detection engine except for providing its input. For instance, to quickly collect the screenshots of all websites, we utilized 125 machines (with 2 cores and 2 GiB memory each), and collection peaked at about 300 screenshots per second. Similarly, once the neural network has been trained, the learned parameters can be distributed to multiple machines and detection can be scaled out horizontally, and, although the system is trained on a GPU, once trained, the detection engine does not require a GPU and can run on common CPUs instead.

Training the system, on the other hand, is not parallelized to multiple machines yet, but some clever tricks can be used to reduce training time significantly [33], which we leave for future work.

3.5 Real-world Deployment

MEERKAT's main deployment is as a monitoring service, acting as an early warning system for website defacements, to which a website operator subscribes with only the URL at which his website can be reached. For each monitored website, the system regularly checks, such as every few minutes (or even seconds), that the website is not defaced. If it detects it as being defaced, it notifies the website's operator, who, in turn, depending on the confidence in the warning, manually investigates, or automatically puts the website in maintenance mode or restores a known good state. Acting as an early warning system, MEERKAT reduces the reaction time to defacements from hours, days, and even weeks (see Section 2) down to minutes (or even seconds), and, therefore, it reduces the damage inflicted on the website's operator by the defacement significantly.

Furthermore, MEERKAT can also reduce human labor: currently, Zone-H manually vets all submissions for defacements [6], of which nearly two thirds are invalid. MEERKAT automates this significant amount of work.

4 Evaluation

We evaluate our implementation of MEERKAT in various settings. However, first, we provide details on what data our dataset is comprised of, and how we partition it to simulate various defacement scenarios.

Our evaluation scenarios are traditional and simulations of real-world events, such as a new defacer group emerging, or how the system's accuracy changes over time, with and without fine-tuning the neural network.

In our experiments, a true positive is a website defacement being detected as a defacement and a true negative is a legitimate website being detected as legitimate. Correspondingly, a false positive is a legitimate website that is being detected as being defaced, and a false negative is a defacement being detected as being legitimate.

4.1 Dataset

The dataset on which we evaluate MEERKAT contains data from two different sources. First, it includes a comprehensive dataset of 10,053,772 defacements observed from January 1998 to May 9, 2014; we obtained this data through a subscription from Zone-H, but it is also freely available from `http://zone-h.org` under a more restrictive license. From those defacements, 9,258,176 defacements were verified manually by Zone-H [6]; the remaining 795,596 website defacements were pending verification and we do not include them in our dataset. Second, our dataset contains 2,554,905 unique (supposedly) undefaced websites from the top 1 million lists from Alexa, MajesticSEO, and QuantCast.[6] Note that we cannot be certain that the legitimate websites in our dataset are not defaced, and since manual verification is impractical at such a large scale, the true negative rate is actually a lower bound and the false positive rate is an upper bound, correspondingly. In layman's terms: the system might be more accurate than our results suggest.[7]

To accurately evaluate the classification performance of MEERKAT in a real-world deployment, we report its accuracy in three different scenarios:

- ❖ Traditional, to compare to prior work, i.e., by performing 10-fold cross-validation by sampling from all data uniformly at random, so that each bin contains 925,817 defacements and 255,490 legitimate websites.
- ❖ Reporter, to simulate a new defacer emerging, i.e., by performing 10-fold cross-validation on the reporters of a defacement and including only their defacements in their respective bin; legitimate website are sampled from the legitimate data uniformly at random.
- ❖ Time-wise, to evaluate the practicality of our approach in a real-world setting, i.e., we start by training the system on all data from December 2012 to December 2013, and, then, we detect defacements from January to May 2014. We report the system's detection accuracy for each month.

We evaluate our system in these settings to prevent a positive skew of our results that might be the result of the different evaluation method and how the dataset is composed. For instance, a reporter of a defacement might introduce an inherit bias to the distribution of the defacement by only reporting the defacements of one specific defacer (such as themselves), or there might be a bias in how defacements and how the web evolved. Those potential pitfalls might skew the results positively or negatively and must be considered for an accurate comparison to prior work.[8]

Finally, to account for the difference in the number of samples of the legitimate websites (2,554,905) and defaced websites (10,053,772), we report the Bayesian detection rate [45]. The Bayesian detection rate is normalized to the number of samples and corresponds to the likelihood if we detect a website as being defaced, it is actually defaced (the likelihood of a positive prediction being correct, that is a true positive; i.e., P(true positive|positive)).

4.2 Features Learned

The features that MEERKAT learns depend on the data it is being trained on. Although one can treat the system as a black-box and not worry about its internal details, understanding how it comes to its final decision helps one to reason about its robustness and to understand how difficult the system is to evade or to estimate when the system must be retrained to retain its accuracy. In our experiments, MEERKAT learned various features automatically and directly from image data, of which we manually grouped some on a higher, more conceptual level together. We manually identified the learned features by evaluating which representative windows activate the same neuron of the neural network, i.e., which windows trigger the same feature to be recognized by MEERKAT. Note that all the features we discuss hereinafter have been learned *automatically* from data and no domain knowledge whatsoever was required to learn and use these features; yet, the overlap with features that an analyst with domain knowledge would use confirms the prospects of feature/representational learning for website defacement detection. Some of the learned features can be best described as:

Defacement group logos. MEERKAT learned to recognize the individual logos of some of the most prolific defacement groups directly (see Figure 4). Clearly, the

[6] We made a list of all 2,554,905 legitimate websites included in our dataset available at `http://cs.ucsb.edu/~kevinbo/sec15-meerkat/legitimate.txt.bz2`.

[7] Over 191,000 website in our legitimate dataset have been defaced at one point in the past, thus, it is likely that some of them are actually defaced and therefore mislabeled; thus, if classified correctly as a defacement by MEERKAT, they appear as false positives in our results.

[8] We cannot compare prior work on our dataset directly as they do not scale to its size, and we cannot compare on their datasets because they are too small to train MEERKAT accurately (see Section 6.1).

logos of the defacer groups themselves are extremely descriptive of website defacements because they are very unlikely to be included in legitimate websites.

Color combinations. MEERKAT also learned to recognize unique or specific color combinations indicative of legitimate and defaced websites, including but not limited to one of the most prominent combinations: bright red or green text on a black background, which is an often used color combination by defacers, but rarely seen on legitimate websites. On the other hand, small black text on a white or brightly colored background is being consulted as a non-definitive indicator for a legitimate, non-defaced website.

Letter combinations. Interestingly, defacers often not only mix colors, but also mix characters from different alphabets right next to each other, such as Arabic or Cyrillic script being mixed with Latin script, to promote their message in both their native language and also in English as the web's *lingua franca*. Additionally, sometimes the defacement contains characters in a character set encoding specific to the defacer's native language, like ISO-8859-13 for Baltic languages or Windows-1256 for Arabic. As such, characters appear differently or are replaced by special characters if the browser does not support it, or if the website does not specify the character set and if the browser's fallback is different (like in our case, as we fall back to UTF-8), resulting in a *look and feel* that is descriptive of defacements, and, correspondingly, it was automatically learned by MEERKAT.

Leetspeak. Similarly to letter combinations, MEERKAT learned that defacers often use "leetspeak," an English alphabet in which some characters are replaced by numbers or special characters (e.g., "leetspeak" as "1337sp34k") and in which some words are deliberately misspelled ("owned" as "pwned," "the" as "teh," or "hax0red" instead of "hacked"). Defacers often use leetspeak to discern themselves from "common folks," and to show that they are "elite" and special, which, in turn, makes it often a good indicator that a website has indeed been defaced.

Typographical and grammatical errors. While some typographical mistakes are deliberate (as in the case of leetspeak, see above), many defacers make other unintentional typographical and grammatical mistakes, which rarely occurred on the legitimate websites in our dataset. Many defacers make these mistakes most likely because they are not native English speakers (the country of the reporter of the defacement, part of the meta-data in our dataset, suggests that most defacers do not speak English as their first language). MEERKAT learned to detect some of these mistakes at training and values them as a supporting indicator of a website defacement. Some of the examples of (supposedly) unintentional typographical and grammatical errors include "greats to" (instead of "greets to"), "goals is" (instead of "goals are"), or "visit us in our website" ("visit us at our website" or just "visit our website").

Note that, since MEERKAT works on image data, the system is unaware that it analyzes text and the textual features, such as unique letter combinations, leetspeak, or typographical and grammatical errors, are actually being evaluated on rendered text. As such, it seems likely that the textual features are specific to the font, possibly overfitting on the specific font type. However, we manually confirmed that the system actually learned a more robust feature and is not overfitting: it combines slight variances in the font family and size in a single high-level feature. Furthermore, given the sliding window approach MEERKAT employs for detection, the features are also completely independent of the position of the text in the representative window and website.

While some of the learned features can be evaded theoretically, evading them almost always contradicts the defacer's goal: making a name for themselves in the most "stylish" and personalized way possible, thus, it is unlikely that these features will change drastically in the near future. Furthermore, MEERKAT also consults features that were not as easy to discern into high-level feature groups manually, such as artifacts unique to legitimate or defaced websites, or features that are indicative for one group but are not definitive because they might appear more often in defaced websites, but also sometimes legitimately. MEERKAT can also be retrained easily and new features are learned *automatically* once the old features do not model defacements accurately anymore (i.e., if the concept of a defacement drifted significantly). Finally, since MEERKAT uses a non-linear classifier to combine those features, it can learn more complex models about defacements and legitimate websites, and simply evading only some features will not be sufficient to evade detection.

Interestingly, some of the high-level features (letter and color combinations) that MEERKAT learned automatically from data have been leveraged to a smaller degree by prior work [46, 47] (through manual feature engineering), while others (logos, leetspeak, and typographical mistakes) had not been utilized yet. Further suggesting that representation learning and inspection of the learned features can yield important insight into security challenges that were dominated by feature engineering in the past, such as intrusion, malware, or phishing detection.

4.3 Traditional Split

First, for an accurate comparison to prior work, we evaluate MEERKAT on our dataset using 10-fold cross-validation, i.e., we split the dataset into 10 bins that contain 925,817 website defacements and 255,490 legitimate websites each. Note that we discard 6 website defacements and 5 legitimate websites from our dataset at random to have the same number of samples in each bin. Next, for each bin, we train the system on the other 9 bins (training bins) and measure its classification performance on the 10^{th} bin (test bin). Considering the 10 different 90% training and 10% test-set partitions of our dataset separately, MEERKAT achieves true positive rates between 97.422% and 98.375%, and false positive rates ranging from 0.547% to 1.419%. The Bayesian detection rate is between 99.603% and 99.845%.

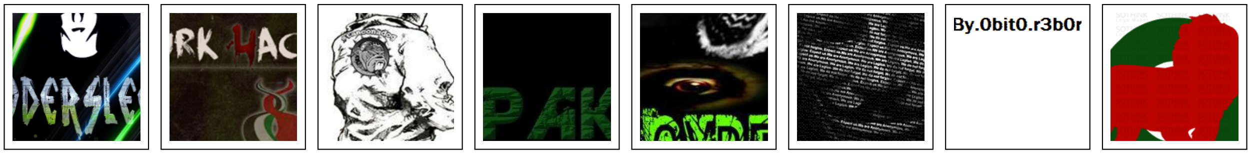


Figure 4: Defacement Group Logos. Example representative windows of logos of defacement groups that MEERKAT learned to recognized to be a significant indicator for defacements. Note that MEERKAT also recognizes variations and that there are many other features used for classification.

More interestingly, as a partition-independent measure of the system's classification performance, the average true positive rate is 97.878%, the average false positive rate is 1.012%, and the average Bayesian detection rate is 99.716%. If MEERKAT detects a defacement and raises an alert, with likelihood 99.716% it is a website defacement. Therefore, MEERKAT is significantly outperforming current state-of-the-art approaches.

4.4 Reporter Split

For the reporter split, we partition our dataset by the reporter of the defaced website. We deliberately designed the experiment this way to show that MEERKAT is not overfitting on specific defacements, which our results verify.

While a partition by reporter might seem counter-intuitive at first, it becomes clear that such a split is meaningful and that it can be used to evaluate that a new defacer group emerges once it is taken into account that these groups often have unique defacement designs and that defaced websites are most often reported by the defacers themselves. Therefore, if we split by reporter, we are practically splitting by defacer group; meaning, we create the most difficult scenario for a defacement detection system: detecting a defacer and his/her defacement style although we have never seen defacements from him/her before.

In the same way as for the traditional split, we employ 10-fold cross-validation. However, we do so slightly differently: first, we separate the reporters of the defacements into 10 bins uniformly at random (each bin containing 7,602 reporters). Second, we construct the corresponding defacement bins, i.e., we construct a defacement bin for each reporter bin so that it contains only the defacements reported by these reporters. For each bin, we then train MEERKAT on the remaining 9 bins and use the 10^{th} bin for testing. Note that the defacement bins contain a different number of samples, simply because the number of reported defacements varies per reporter (see Appendix A). We account for the uneven distribution of defacements by reporting the average true positive and false positive rate weighted by the number of samples.

Overall, when simulating the emergence of a new defacer, MEERKAT achieves a true positive rate of 97.882% and a false positive rate of 1.528% if bins are weighted, and 97.933% and 1.546% if they are not (see Figure 5; the true positive rate is between 97.061% and 98.465%, the false positive rate is between 0.661% and 2.564%). The Bayesian detection rates for the reporter split are 99.567% (unweighted) and 99.571% (weighted) respectively (per split, the Bayesian detection rate is between 99.286% and 99.814%).

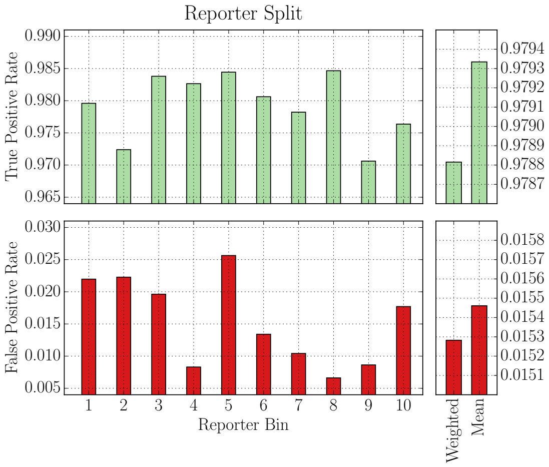


Figure 5: True positive and false positive rates for the reporter split, per bin of the 10-fold cross-validation set. Note that the scales for true positives and false negatives are the same, but that the y-axis goes from 0.965 to 0.99 for the true positive rate and 0.005 to 0.03 for the false positive rate. The weighted mean true positive rate is 97.882% and its false positive rate is 1.528% (weighted by samples per bin). The unweighted mean true positive rate is 97.933% and its false positive rate is 1.546%.

4.5 Time-wise Split

The time-wise experiment evaluates how well MEERKAT detects website defacements in the wild, i.e., in a real-world deployment. Here, we train the system on defacements seen in the past, and we detect defacements in the present. Similarly to the reporter split, the time-wise experiment shows that MEERKAT does not overfit on past defacements, and that it successfully detects present defacements.

Our training set selection follows a simple argument: it is extremely unlikely that websites today will be defaced in the same way as they were defaced in 2005 or even 1998. Including those defacements in our training set would then very likely decrease classification performance for defacement detection in 2014. Equivalently, one would not include this data to train the system in practice.

We train MEERKAT on all defacements that were reported between December 2012 and December 2013 (including, i.e., 13 months with 1,778,660 defacements observed in total), and 1,762,966 legitimate websites that we sample from all legitimate websites uniformly at random. We then detect defacements over a five months time frame, from January to May 2014, and we report the classification performance for each month. The test data from January to May 2014 spans a total of 1,538,878 unique samples that are distributed as follows: 421,758 samples from January 2014, 364,168 samples from February 2014, 474,758 samples from March 2014, 241,926 samples from April 2014, and 81,268 samples from the beginning of May 2014.

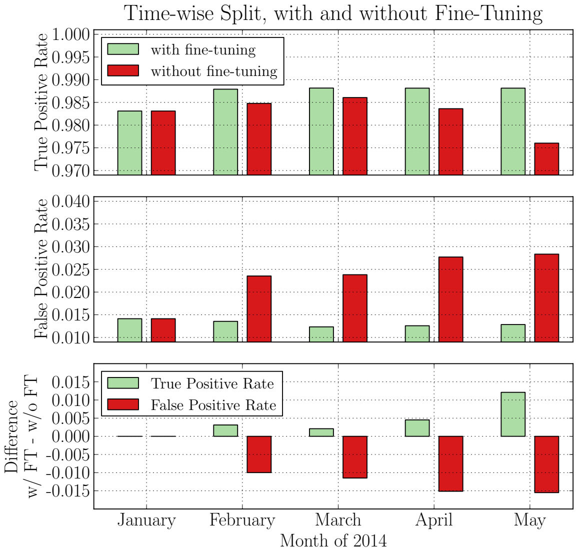


Figure 6: True positive and false positive rates, and the difference with and without fine-tuning, for the time-wise split. Note that the scales for true positives and false negatives are the same, but that the y-axis goes from 0.97 to 1 for the true positive rate and 0.01 to 0.04 for the false positive rate. No significant change is visible for the true positive rate in the beginning regardless if the network is fine-tuned regularly or not, however, a non-negligible difference is observable for May 2014. A difference is observable for the false positive rate starting in February 2014, after the network was first fine-tuned.

In detail, MEERKAT achieves a true positive rate between 98.310% and 98.816% when the system is fine-tuned after each month on the data observed in that month, and 97.603% to 98.606% when it is not. Although there is no significant difference in its accuracy from January to March when the neural network is fine-tuned and when it is not (see Figure 6), a non-negligible difference between their accuracy can be observed for April and the beginning of May (increase in 0.452 percentage points (pp) and 1.211 pp for the true positive rate; decrease of 1.513 pp and 1.550 pp for the false positive rate). The Bayesian detection rate if no fine-tuning is used decreases from 98.583% in January 2014 to 97.666% in February (0.917 pp decrease) to 97.177% in May (1.406 pp decrease to January). If fine-tuning is utilized, the Bayesian detection rate increases from 98.583% in January 2014 to 98.717% in May (0.134 pp).

Unsurprisingly, the regularly fine-tuned system performs better over time, probably because some defacers became significantly more active in 2014, like *Team System Dz*, who started to deface websites just in January 2014 and who were not active before at all, and because some defacers changed their defacements to spread a different message as opposed to the one they spread the year before. When the system is not fine-tuned, however, these minor changes to the defacements allow attackers to evade detection without actively trying to evade it, with a minor accuracy deterioration already visible after just four to five months, confirming that detection systems need to be able to tackle even minor concept drift adequately and gracefully to maintain accurate detection capabilities over time, like MEERKAT does with fine-tuning.

5 Limitations

Similar to other systems leveraging machine learning, our system has some limitations that can be used to evade detection. We discuss some of these limitations and show how they can be addressed for a real-world deployment. First, we discuss *concept drift*, a problem all systems leveraging machine learning have to deal with; second, we remark on *browser fingerprinting* and *delayed defacement*, an issue all client-based detection approaches have to address; and, lastly, we introduce the concept of *tiny defacements*, a limitation specific to defacement detection systems.

5.1 Concept Drift

Concept drift is the problem of predictive analysis approaches, such as detection systems, that the statistical properties of the input used to train the models change. In turn, a direct result of concept drift is often a heavy deterioration of the classification performance, up to the point where the system cannot differentiate between good and bad behavior anymore. For instance, prior work [48–55] has shown that concept drift (in the sense of adversarial learning) can actually be leveraged to evade detection systems and classifiers in practice. Therefore, a detection system must address it.

While concept drift is a major issue for all systems using machine learning, it can generally be addressed, due to its nature, by adopting a new feature space or retraining the machine learning model on new data, or with an increased weight on new data. However, often, old instances do not follow the statistical properties of the new feature models, and, therefore, they are classified less accurately than before. This has little impact in practice, because old instances are less likely to occur in the future anyways; yet, it is important to realize that this approach allows attackers to evade the system by oscillating their attack strategy regularly.

For MEERKAT, those shortcomings can be addressed more easily than for traditional systems: for minor concept drift, the system's accuracy can be maintained by fine-tuning the parameters of the network. Here, the system simply needs to learn minor adjustments to the weights of existing features from new data, because some features have become more important and others have become less important (they differ now more from other features than they did previously, relatively speaking; since we start with already-initialized weights, fine-tuning requires much less time than training the whole system again). Here, the features still model the differences between defacements and legitimate websites, however, the weights are not optimal anymore and need to be adjusted. Once the new weights are learned, classification performance is restored. Therefore, to address minor concept drift adequately, we recommend fine-tuning the model regularly, e.g., every month (see Section 4.5).

While fine-tuning the system's parameters can theoretically address major concept drift similar to retraining the system on new data, in practice, we expect prediction accuracy to decrease, since different or more features must be modeled with the same amount of resources. Instead, for major concept drift, increasing the number of hidden nodes of the neural network that learn the compressed representation (the features) and their weights, and then retraining the system

Split	True Positive Rate	False Positive Rate	Bayesian Detection Rate
Traditional	97.878%	1.012%	99.716%
Reporter (weighted)	97.882%	1.528%	99.571%
Reporter (unweighted)	97.933%	1.546%	99.567%
Time-wise with fine-tuning	98.310% - 98.816%	1.233% - 1.413%	98.583% - 98.767%
Time-wise without fine-tuning	97.603% - 98.606%	1.413% - 2.835%	97.177% - 98.583%

Table 2: Average true positive, false positive, and Bayesian detection rates for traditional and reporter split. Lower and upper bound of true positive, false positive, and Bayesian detection rate for time-wise split from January to May 2014.

can maintain the system's accuracy. Simply adding nodes to the hidden layers of the neural network can counteract the issue of major concept drift because we increase the number of features that MEERKAT learns from data directly. Therefore, introducing more hidden units allows the system to learn additional and different internal representations about the *look and feel* of defacements, while, at the same time, maintaining a model of how the old defacements look like. However, it requires computationally-costly retraining of the network (previously, having those additional hidden units in the network would result in overfitting because the system would learn more complex representations than necessary, and each would only differ little from one another; the system would then be prone to missing minor variations of defacements).

It is important to note that in both cases, for minor and major concept drift, MEERKAT requires no additional feature engineering because the features are learned *automatically* from data. In turn, this allows MEERKAT to handle any form of concept drift much more gracefully than approaches introduced by prior approaches, which require manual feature engineering.

5.2 Fingerprinting and Delayed Defacement

A second limitation of detection systems is fingerprinting. Since we are leveraging a web browser to collect the data that we are analyzing, in our case fingerprinting corresponds to IP-based and browser fingerprinting. For IP-based fingerprinting, a set of VPNs and proxies can be used to cloak and regularly change the browser's IP address. In case of browser fingerprinting, the server or some client-side JavaScript code detects what browser is rendering the website, and then displays the website differently for different browsers. In its current form, the screenshot engine from MEERKAT might be detectable (to some degree) by browser fingerprinting. It is theoretically possible to detect it because it is currently built on the headless browser PhantomJS rather than a "headful" browser typically used by a normal user, like Google Chrome. However, since PhantomJS is built from the same components as Google Chrome, fingerprinting the current screenshot engine is not trivial and requires intimate knowledge of the differences between the different versions of the components and their interaction. Therefore, we argue that the evasion through browser fingerprinting is unlikely. If, however, the screenshot engine is evaded this way in the future, only some minor engineering effort is required to utilize a browser extension to retrieve the websites' screenshots instead.[9]

Additionally, the issue of delaying the defacement emerges, also referred to as the snapshot problem [30]. With the increased popularity and use of JavaScript, client-side rendering, and asynchronous requests to backends by websites to provide a seamless and "reload-free" user experience, it is uncertain at what point in time a website is representative of how a user would experience it. This then bears the issue of when a detection system can take a representative snapshot of the website and stop executing client-side scripts. For instance, if a detection system takes a snapshot always after five seconds, to evade detection, defacers could simply inject JavaScript that only defaces the website if a user interacts with it for at least six seconds.

While delayed defacements are currently scarce, it is likely that they will gain some traction once more detection systems have been put in place, in a way similar to mimicry attacks and the evasions of malware detection systems [56, 57]. However, prior work can be leveraged to detect evasions [58] or trigger the functionality [59] to force the defacement to be shown. Both approaches are complementary to MEERKAT and we leave their adoption to defacement detection for future work, once delayed defacements are actually occurring in the wild.

5.3 Tiny Defacements

A third limitation of all current defacement detection systems, including MEERKAT, is the lack of detection capabilities for tiny defacements. Tiny defacements describe a class of defacements in which only a very minor modification is made to part of the content of the defaced website. For instance, a defacer might be dissatisfied by an article published by a newspaper. Instead of defacing the website as a whole, the attacker modifies (or deletes) the news article. It is clear that such defacements are very hard to differentiate from the original content because they might only have minor semantic changes to text or images. Thus, to detect tiny defacements, the detection system must understand the semantics of the website's content, its language, and its general behavior to derive a meaningful model for the website.

However, while those defacements exist, they are extremely scarce in numbers, or they are rarely noticed. In fact, it is seldom the case that a defacer wants to modify a website without embarrassing the operator more publicly. Most often, the goal of the defacer is to expose the insecurity

[9] In fact, we are migrating our screenshot engine to Chrome, eliminating the problem that PhantomJS might be fingerprinted.

of the website, ridicule the operator publicly, show their own "superiority," and place their opinion and beliefs in the most public space possible. Therefore, tiny defacements are currently of little interest to the defacers themselves, and, hence, also of little impact for detection systems. However, we acknowledge that tiny defacements must be addressed once they increase in numbers, possibly leveraging recent work to extract relevant changes from websites [60], and advances in natural language processing.

6 Related Work

Hereinafter, we discuss related work that detects defacement, and image-based detection used in computer security.

6.1 Defacement Detection

Similar to MEERKAT, Davanzo et al. [46] introduce a system that acts a monitoring service for website defacements. Their system utilizes the website's HTML source code for detection, and its features were selected manually based on domain knowledge acquired *a priori*, making the system prone to concept drift. On their, comparatively, very small dataset containing only 300 legitimate websites and 320 defacements, they achieve false positive rates ranging from 3.56% to 100% (depending on the machine learning algorithm used; suggesting extreme under- and over-fitting with some algorithms), and true positive rates between 70.07% to 100% (in the case of simply classifying everything as a defacement; i.e., extremely under-fitting the dataset). Overall, these results are significantly worse than MEERKAT, both in terms of false positives (1.012%) and true positives (97.878%).

Bartoli et al. [47] propose Goldrake, a website defacement monitoring tool that is very similar to the tool proposed by Davanzo et al. and leverages a superset of their features. To learn an accurate model, Goldrake requires knowledge about the monitored website to learn website-specific parameters. However, it is unclear how well Goldrake detects defacements in practice because it is evaluated on a small and (likely) non-diverse dataset comprised of only 11 legitimate websites and 20 defacements, on which it performs poorly with a high false negative rate (27%).

Medvet et al. [61] introduce a defacement detection system based on work by Bartoli et al. and Davanzo et al., but the detection engine is replaced by a set of functions that are learned through genetic programming. The learned functions take the features by Bartoli et al. and Davanzo et al. as input, but classification is more accurate on a dataset comprised of 15 websites (between 0.71% and 23.38% false positives, and about 97.52% true positives). It is, again, unclear how the system would fare in a real-world deployment because of the small and (likely) non-diverse dataset.

Note that all text-based approaches have major shortcomings, similar as those encountered in spam and phishing detection, such as using images to show text to evade detection. MEERKAT does not suffer from these shortcomings.

Lastly, most commercial products that detect website defacements are built upon host-based intrusion detection systems to monitor modifications of the files on the web server, e.g. via file integrity checks (checksums) [62, 63]. Therefore, those approaches bear the major shortcoming that they can only detect the subset of defacements that modify files on disk, and that they cannot detect other defacement attacks, such as through SQL injections; even when the defacements look exactly the same to the website's visitors. MEERKAT does not suffer from this shortcoming.

6.2 Image-based Detection in Security

Since, to the best of our knowledge, no prior work applies image-based methods to detect defacements, we compare prior work to defacement detection that visually detects phishing pages, or leverages image-based techniques as part of a larger system.

Medvet et al. [64] propose a system to detect if a potential phishing page is similar to a legitimate website. The system leverages features such as parts of the visible text, the images embedded in the website, and the overall appearance of the website as rendered by the browser for detection. Similarity is measured by comparing the 2-dimensional Haar wavelet transformations of the screenshots. Their system achieves a 92.6% true positive rate and a 0% false positive rate on a dataset comprised of 41 real-world phishing pages.

Similarly, Liu et al. [65] present an antiphishing solution that is deployed at an email server and detects linked phishing pages by assessing the visual similarity to the legitimate page, but only when analysis is triggered on keyword detection. The system detects phishing pages by comparing the suspicious website to the legitimate website by measuring similarity between text and image properties, like the font size and family used, or source of an image.

While detecting phishing pages by comparing the similarity of two websites makes sense, for defacements the difference between them is more interesting. Instead of creating a visually-similar page to trick users into submitting their credentials, a defacer wants to promote his message. Adopting existing phishing detection systems to detect defacements instead, i.e., by comparing if the website looks different from its usual representation, however, bears two problems: (a) the usual representation must be known and/or stored for comparison, and (b) false positives are much more likely if the website is dynamic or if it shows regularly-changing ads.

Anderson et al. [29] introduce image shingling, a technique similar to w-shingling, to cluster screenshots of scams into campaigns. However, in its current form, image shingling cannot be used to detect defacements as it is trivial to evade the clustering step with only minor modifications that are invisible to the human eye, and, thus, the technique is unsuitable for a detection system in an adversarial context.[10]

Nappa et al. [66] leverage perceptual hashing to group visually similar icons of malicious executables under the assumption that a similar icon suggests that the two executables are part of the same malware distribution campaign. While it is theoretically possible to detect defacements through perceptual hashing-based techniques and comparing the distance of the hashes, it is impractical to do so on a large scale and in an adversarial context. For

[10] The authors acknowledge the shortcomings in an adversarial context in Section 4.2, but they do not discuss any remediation techniques.

once, one must have a ground-truth screenshot that is close enough to the screenshot that one wants to classify; if ground-truth is not available or slightly too different, a system based on perceptual hashing will be unable to detect the defacement. Furthermore, classification is not constant in the number of defacements the system has seen in the past: for each new screenshot we would want to classify, we would need to compute the distance to the hashes of at least some (or all) of the previously-seen defacements.[11]

Grier et al. [67] introduce their own image similarity measure to cluster malicious executables that have similar looking user-interface components after being executed in a dynamic analysis environment. Two images are considered similar if the root mean squared deviation between the images' histograms is below some manually-determined threshold. Clearly, a defacement system based on this technique is not suitable in an adversarial context: an attacker can (and eventually will) simply change the colors slightly or add dynamic content, so that the root mean squared deviation is above the threshold, but remains visually the same to the human eye. Furthermore, exactly as for Nappa et al. [66], one needs to pair-wise compare the histogram of the screenshot one wants to classify to some or all of the already-seen defacements.[11]

MEERKAT does not suffer from these shortcomings: first, it learns high-level features on the defacements' general *look and feel* to detect also previously unseen defacements, and, second, its classification time is constant in the number of already-seen defacements.

7 Conclusions

In this paper, we introduced MEERKAT, a monitoring system to detect website defacements, which utilizes a novel approach based on the *look and feel* of a website to identify if the website has been defaced. To accurately identify website defacements, MEERKAT leverages recent advances in machine learning, like stacked autoencoders and deep neural networks, and combines them with computer vision techniques. Different from prior work, MEERKAT does not rely on additional information supplied by the website's operator, or on manually-engineered features based on domain knowledge acquired *a priori*, such as how defacements look. Instead, MEERKAT automatically learns high-level features from data directly. By deciding if a website has been defaced based on a region of the screenshot of the website instead of the whole screenshot, the system is robust to the normal evolution of websites and defacements and can be used at scale. Additionally, to prevent the evasion of the system through changes to the *look and feel* of defacements and to be robust against defacement variants, MEERKAT employs various techniques, such as dropout and fine-tuning.

We showed the practicality of MEERKAT on the largest website defacement dataset to date, spanning 10,053,772 defacements observed between January 1998 and May 2014, and 2,554,905 legitimate websites. On this dataset, in different scenarios, the system accurately detects defacements with a true positive rate between 97.422% and 98.816%, a false positive rate between 0.547% and 1.528%, and a Bayesian detection rate between 98.583% and 99.845%, thus significantly outperforming existing state-of-the-art approaches.

8 Acknowledgments

We want to express our gratitude toward the reviewers for their helpful feedback, valuable comments and suggestions to improve the quality of the paper.

This work was supported by the Office of Naval Research (ONR) under grant N00014-12-1-0165, the Army Research Office (ARO) under grant W911NF-09-1-0553, the Department of Homeland Security (DHS) under grant 2009-ST-061-CI0001, the National Science Foundation (NSF) under grant CNS-1408632, Lastline Inc., and SBA Research.

References

[1] G. Davanzo, E. Medvet, and A. Bartoli, "A Comparative Study of Anomaly Detection Techniques in Web Site Defacement Detection", in *Proceedings of the IFIP 20th World Computer Congress*, Springer, 2008.

[2] Anonymous, *Reference blinded for double-blind review process*, Nov. 2014. [Online]. Available: `http://anonymized`.

[3] Wall Street Journal (WSJ), *Malaysia Airlines Website Hacked by Group Calling Itself 'Cyber Caliphate'*, Jan. 26, 2015. [Online]. Available: `http://goo.gl/RhO2tO`.

[4] British Broadcasting Company (BBC), *Keighley Cougars website hacked to read 'I love you Isis'*, Nov. 2014. [Online]. Available: `http://goo.gl/bzxJ8M`.

[5] R. Preatoni, M. Almeida, K. Fernandez, and other unknown authors, *Zone-H.org - Unrestricted Information*, since January 1998. [Online]. Available: `http://zone-h.org/`.

[6] E. Kovacs, *Softpedia Interview: Alberto Redi, Head of Zone-H*, Jun. 8, 2013. [Online]. Available: `http://goo.gl/cwPBrW`.

[7] Malaysian Computer Emergency Response Team, *MyCERT Incident Statistics*, Jan. 2014. [Online]. Available: `http://goo.gl/0LTRPj`.

[8] CyberSecurity Malaysia, "MyCERT 2nd Quarter 2013 Summary Report", *eSecurity Bulletin*, vol. 34, Aug. 2013.

[9] S. Mansfield-Devine, "Hacktivism: assessing the damage", *Network Security*, vol. 2011, no. 8, 2011.

[10] M. Gorge, "Cyberterrorism: hype or reality?", *Computer Fraud & Security*, vol. 2007, no. 2, 2007.

[11] H. Kircher, "The Practice of War: Production, Reproduction and Communication of Armed Violence", in. Berghahn Books, Mar. 2011, ch. 12. Martyrs, Victims, Friends and Foes: Internet Representations by Palestinian Islamists.

[12] G. Weimann, "Terror on the Internet: The New Arena, the New Challenges", in. US Institute of Peace Press, 2006, ch. 6. Fighting Back: Responses to Terrorism on the Internet, and Their Cost.

[13] Wall Street Journal (WSJ), *Google Access Is Disrupted in Vietnam*, Feb. 23, 2015. [Online]. Available: `http://goo.gl/JlVtfW`.

[14] L. Makani, *100+ Zambian websites hacked & defaced: Spar, Postdotnet, SEC, Home Affairs, Ministry of Finance*, Apr. 2014. [Online]. Available: `http://goo.gl/NvQsJM`.

[15] British Broadcasting Company (BBC), *Angry Birds website hacked after NSA-GCHQ leaks*, Jan. 2014. [Online]. Available: `http://goo.gl/kHDIAj`.

[11] Detection time increases with each observed defacement; it is at best in $O(\log n)$ and at worst in $O(n)$, with n being all observed defacements.

[16] A. Mittal, *NIC of Suriname, Antigua & Barbuda and Saint Lucia Hacked by Pakistani Hackers*, Oct. 2013. [Online]. Available: `http://goo.gl/ynGG0y`.

[17] J. Leyden, *Islamist hackers attack Danish sites*, Feb. 2006. [Online]. Available: `http://goo.gl/jcE7iv`.

[18] ——, *Hacktivists attack UN.org*, Aug. 2007. [Online]. Available: `http://goo.gl/SfvkUc`.

[19] G. Maone, *United Nations vs. SQL Injections,* Aug. 2007. [Online]. Available: `http://goo.gl/v8oXih`.

[20] S. Reid, *Hip-Hop Sites Hacked By Apparent Hate Group; SOHH, AllHipHop Temporarily Suspend Access*, Jun. 2008. [Online]. Available: `http://goo.gl/VtW4i6`.

[21] B. Acohido, *State Department webpages defaced*, Oct. 23, 2013. [Online]. Available: `http://goo.gl/698XRW`.

[22] J. Leyden, *Foxconn website defaced after iPhone assembly plant suicides*, May 2010. [Online]. Available: `http://goo.gl/6BtZbX`.

[23] ——, *Anti-Israel hackers deface central bank site*, Apr. 2008. [Online]. Available: `http://goo.gl/7Ve2xT`.

[24] British Broadcasting Company (BBC), *Nottinghamshire Police website hacked by AnonGhost*, Nov. 2014. [Online]. Available: `http://goo.gl/Gbldxt`.

[25] ——, *Shropshire Fire Service website hacked by AnonGhost*, Nov. 2014. [Online]. Available: `http://goo.gl/3dq4Cq`.

[26] D. Dagon, M. Antonakakis, P. Vixie, T. Jinmei, and W. Lee, "Increased DNS Forgery Resistance Through 0x20-Bit Encoding: SecURItY viA LeET QueRieS", in *Proceedings of the 15th ACM Conference on Computer and Communications Security (CCS)*, ACM, 2008.

[27] G. Vigna and C. Kruegel, "Host-based Intrusion Detection", *Handbook of Information Security. John Wiley and Sons*, 2005.

[28] A. Bartoli, G. Davanzo, and E. Medvet, "The Reaction Time to Web Site Defacements", *Internet Computing, IEEE*, vol. 13, no. 4, 2009.

[29] D. S. Anderson, C. Fleizach, S. Savage, and G. M. Voelker, "Spamscatter: Characterizing Internet Scam Hosting Infrastructure", in *Proceedings of 16th USENIX Security Symposium on USENIX Security Symposium*, ser. SS'07, USENIX Association, 2007.

[30] K. Borgolte, C. Kruegel, and G. Vigna, "Delta: Automatic Identification of Unknown Web-based Infection Campaigns", in *Proceedings of the 20th ACM SIGSAC Conference on Computer and Communications Security (CCS)*, ACM, 2013.

[31] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun, "OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks", in *Proceedings of the 2nd International Conference on Learning Representations (ICLR)*, CBLS, Apr. 2014.

[32] Q. Le, M. Ranzato, R. Monga, M. Devin, K. Chen, G. Corrado, J. Dean, and A. Ng, "Building High-level Features Using Large Scale Unsupervised Learning", in *Proceedings of the 29th International Conference on Machine Learning (ICML)*, IMLS, Jun. 2012.

[33] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks.", in *Advances in Neural Information Processing Systems 25 (NIPS)*, vol. 1, 2012.

[34] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation", *arXiv preprint arXiv:1311.2524*, 2013.

[35] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition", *Proceedings of the IEEE*, vol. 86, no. 11, 1998.

[36] R. Raina, A. Madhavan, and A. Y. Ng, "Large-scale deep unsupervised learning using graphics processors", in *Proceedings of the 26th International Conference on Machine Learning (ICML)*, 2009.

[37] Q. V. Le, J. Ngiam, Z. Chen, D. J. hao Chia, P. W. Koh, A. Y. Ng, and D. Chia, "Tiled convolutional neural networks.", in *Advances in Neural Information Processing Systems 23 (NIPS)*, 2010.

[38] H. Lee, R. Grosse, R. Ranganath, and A. Y. Ng, "Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations", in *Proceedings of the 26th Annual International Conference on Machine Learning (ICML)*, ACM, 2009.

[39] P. Sermanet, S. Chintala, and Y. LeCun, "Convolutional neural networks applied to house numbers digit classification", in *Proceedings of the 21st International Conference on Pattern Recognition (ICPR)*, IEEE, 2012.

[40] A. Hyvärinen, J. Hurri, and P. O. Hoyer, *Natural Image Statistics: A Probabilistic Approach to Early Computational Vision.* Springer, 2009, vol. 39.

[41] K. Gregor and Y. LeCun, "Emergence of complex-like cells in a temporal product network with local receptive fields", *arXiv preprint arXiv:1006.0448*, 2010.

[42] K. Jarrett, K. Kavukcuoglu, M. Ranzato, and Y. LeCun, "What is the best multi-stage architecture for object recognition?", in *Proceedings of the 12th IEEE International Conference on Computer Vision*, IEEE, 2009.

[43] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors", *arXiv preprint arXiv:1207.0580*, 2012.

[44] Y. Jia, *Caffe: An Open Source Convolutional Architecture for Fast Feature Embedding*, 2013. [Online]. Available: `http://goo.gl/Fo9YO8`.

[45] S. Axelsson, "The Base-Rate Fallacy and the Difficulty of Intrusion Detection", *ACM Transactions on Information and System Security (TISSEC)*, vol. 3, no. 3, 2000.

[46] G. Davanzo, E. Medvet, and A. Bartoli, "Anomaly Detection Techniques for a Web Defacement Monitoring Service", *Expert Systems with Applications*, vol. 38, no. 10, 2011.

[47] A. Bartoli and E. Medvet, "Automatic Integrity Checks for Remote Web Resources", *Internet Computing, IEEE*, vol. 10, no. 6, 2006.

[48] L. Huang, A. D. Joseph, B. Nelson, B. I. Rubinstein, and J. Tygar, "Adversarial Machine Learning", in *Proceedings of the 4th ACM Workshop on Security and Artificial Intelligence (AISEC)*, ACM, Oct. 2011.

[49] M. Barreno, B. Nelson, A. D. Joseph, and J. Tygar, "The Security of Machine Learning", *Machine Learning*, vol. 81, no. 2, 2010.

[50] M. Barreno, B. Nelson, R. Sears, A. D. Joseph, and J. D. Tygar, "Can machine learning be secure?", in *Proceedings of the 13th ACM Symposium on Information, Computer and Communications Security (CCS)*, ACM, Oct. 2006.

[51] N. Šrndic and P. Laskov, "Practical Evasion of a Learning-Based Classifier: A Case Study", in *Proceedings of the 35th IEEE Symposium on Security and Privacy (Oakland)*, IEEE, May 2014.

[52] D. Lowd and C. Meek, "Adversarial Learning", in *Proceedings of the 11th ACM SIGKDD International Conference on Knowledge Discovery in Data Mining (KDD)*, ACM, Aug. 2005.

[53] N. Dalvi, P. Domingos, Mausam, S. Sanghai, and D. Verma, "Adversarial Classification", in *Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, ACM, 2004.

[54] A. Globerson and S. Roweis, “Nightmare at Test Time: Robust Learning by Feature Deletion”, in *Proceedings of the 23rd International Conference on Machine Learning (ICML)*, ACM, 2006.

[55] H. Xiao, H. Xiao, and C. Eckert, “Adversarial label flips attack on support vector machines”, in *Proceedings of the 20th European Conference on Artificial Intelligence (ECAI)*, Aug. 2012.

[56] D. Wagner and P. Soto, “Mimicry Attacks on Host-based Intrusion Detection Systems”, in *Proceedings of the 9th ACM Conference on Computer and Communications Security (CCS)*, ACM, 2002.

[57] C. Kruegel, E. Kirda, D. Mutz, W. Robertson, and G. Vigna, “Automating Mimicry Attacks Using Static Binary Analysis”, in *Proceedings of the 14th Conference on USENIX Security Symposium*, USENIX Association, 2005.

[58] A. Kapravelos, Y. Shoshitaishvili, M. Cova, C. Kruegel, and G. Vigna, “Revolver: An Automated Approach to the Detection of Evasive Web-based Malware”, in *Proceedings of the 22nd USENIX Security Symposium*, 2013.

[59] C. Kolbitsch, E. Kirda, and C. Kruegel, “The Power of Procrastination: Detection and Mitigation of Execution-stalling Malicious Code”, in *Proceedings of the 18th ACM Conference on Computer and Communications Security (CCS)*, ACM, 2011.

[60] K. Borgolte, C. Kruegel, and G. Vigna, “Relevant Change Detection: Framework for the Precise Extraction of Modified and Novel Web-based Content as a Filtering Technique for Analysis Engines”, in *Proceedings of the Companion Publication of the 23rd International World Wide Web Conference (WWW)*, IW3C2, 2014.

[61] E. Medvet, C. Fillon, and A. Bartoli, “Detection of Web Defacements by Means of Genetic Programming”, in *Proceedings of the 3rd International Symposium on Information Assurance and Security*, IEEE Computer Society, 2007.

[62] G. H. Kim and E. H. Spafford, “The Design and Implementation of Tripwire: A File System Integrity Checker”, in *Proceedings of the 2nd ACM Conference on Computer and Communications Security (CCS)*, ACM, 1994.

[63] A. G. Pennington, J. D. Strunk, J. L. Griffin, C. A. N. Soules, G. R. Goodson, and G. R. Ganger, “Storage-based Intrusion Detection: Watching Storage Activity for Suspicious Behavior”, in *Proceedings of the 12th Conference on USENIX Security Symposium*, USENIX Association, 2003.

[64] E. Medvet, E. Kirda, and C. Kruegel, “Visual-similarity-based Phishing Detection”, in *Proceedings of the 4th International Conference on Security and Privacy in Communication Networks (SecureComm)*, ACM, 2008.

[65] W. Liu, X. Deng, G. Huang, and A. Y. Fu, “An Antiphishing Strategy Based on Visual Similarity Assessment”, *Internet Computing, IEEE*, vol. 10, no. 2, 2006.

[66] A. Nappa, M. Rafique, and J. Caballero, “Driving in the Cloud: An Analysis of Drive-by Download Operations and Abuse Reporting”, English, in *Detection of Intrusions and Malware, and Vulnerability Assessment*, ser. Lecture Notes in Computer Science, K. Rieck, P. Stewin, and J.-P. Seifert, Eds., vol. 7967, Springer Berlin Heidelberg, 2013. [Online]. Available: `http://goo.gl/Z2IJ4D`.

[67] C. Grier, L. Ballard, J. Caballero, N. Chachra, C. J. Dietrich, K. Levchenko, P. Mavrommatis, D. McCoy, A. Nappa, A. Pitsillidis, N. Provos, M. Z. Rafique, M. A. Rajab, C. Rossow, K. Thomas, V. Paxson, S. Savage, and G. M. Voelker, “Manufacturing Compromise: The Emergence of Exploit-as-a-Service”, in *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, ser. CCS ’12, ACM, 2012. [Online]. Available: `http://goo.gl/M1DOdZ`.

Appendix

A Reporter Cross-validation Split

In our reporter split experiment (Section 4.4), we split the dataset by reporter to simulate that a new defacer group emerges. Each cross-validation bin contains the same amount of reporters, but because they reported different numbers of defacements, bins do not contain the same amount of samples. We account for the size difference in our experiments by weighting each bin. Table 3 lists the number of samples per bin.

Bin	Defacements	Legitimate Websites
1	1,116,808	308,202
2	992,232	273,823
3	712,270	196,563
4	907,306	250,387
5	696,069	192,092
6	734,208	202,617
7	1,276,764	352,345
8	789,895	217,985
9	979,309	270,257
10	1,053,147	290,634
Total	9,258,008	2,554,905

Table 3: Number of samples per cross-validation bins used for the reporter split. Note that the total number of defacements in the reporter split contains 168 defacements less than available in the whole dataset because otherwise reporters would be distributed unevenly per bin. However, due to the considerable size of the dataset, omitting these defacements has negligible impact.

B Image-based Object Recognition

Much prior work has been carried out in computer vision to classify images and recognize objects in images. Most recently, object recognition underwent a “new spring” with the rise of deep learning. Deep learning gained traction because training them on large datasets became computationally feasible, and they consistently outperformed other algorithms. We discuss our two main inspirations.

Le et al. [32] introduce a feature learning approach that leverages unsupervised learning with a deep networks comprised of stacked sparse autoencoders utilizing pooling and local contrast normalization. The main idea is to learn high-level features from only unlabeled data (10 million pictures from random Youtube videos); high-level features such as if the image contains a cat, or a human face or body part. After training, the network improves relatively to prior state-of-the-art by 70% on the ImageNet dataset.

Krizhevsky et al. [33] employed supervised learning to train a deep convolutional neural network to classify 1.2 million images spanning 1,000 classes from a subset of the ImageNet dataset and they improve considerably on the state-of-the-art with a top-1 error rate of 37.5% (the classifier is correct for 62.5%) and a top-5 error of 17.0% (for 83% images, the correct class is among top 5 classes). To not overfit the dataset and to reduce the network’s training time, they use rectified linear units as the neurons’ output functions.