Eyes of a Human, Eyes of a Program: Leveraging Different Views of the Web for Analysis and Detection

Jacopo Corbetta, Luca Invernizzi, Christopher Kruegel, and Giovanni Vigna

University of California, Santa Barbara
{jacopo,invernizzi,chris,vigna}@cs.ucsb.edu

Abstract. With JavaScript and images at their disposal, web authors can create content that is immediately understandable to a person, but is beyond the direct analysis capability of computer programs, including security tools. Conversely, information can be deceiving for humans even if unable to fool a program.
In this paper, we explore the discrepancies between user perception and program perception, using content obfuscation and counterfeit "seal" images as two simple but representative case studies. In a dataset of 149,700 pages we found that benign pages rarely engage in these practices, while uncovering hundreds of malicious pages that would be missed by traditional malware detectors.
We envision that this type of heuristics could be a valuable addition to existing detection systems. To show this, we have implemented a proof-of-concept detector that, based solely on a similarity score computed on our metrics, can already achieve a high precision (95%) and a good recall (73%).

Keywords: website analysis · content obfuscation · fraud detection

1 Introduction

Web pages available on the Internet are visited by two very different kinds of consumers: humans, surfing through their web browsers, and computer programs, such as search engines.

These consumers have dissimilar goals and constraints, which lead to significant differences in how they interpret and interact with web pages. For example, a large-scale web crawler, optimized for speed, may not run JavaScript, and thus will not capture dynamically-rendered content: To overcome this issue, search engines have established practices [13, 14, 28, 47] that web authors should follow to make content accessible to their non-JavaScript-aware crawlers. In short, (at least) two different views exist for each page, and search engines rely on web authors to "bridge" the two worlds and make sure a human and a crawler "see" an equivalent message.

Search engines, however, have a privileged role online: Successful web sites need their pages to be indexed, so that people can find them easily; therefore, web

authors are highly encouraged to tailor content so that it is easily consumable by these programs. On the contrary, tools that look for cybercrime activity do not benefit from their role, as they have to face web authors that are always looking for new ways to evade detection. However, even benign web authors can present visual information to humans that is, intentionally or not, hard to digest for crawling tools, for instance by showing text through a combination of images and JavaScript code, a Turing-complete language. In the strictest sense, therefore, only programs with human-like comprehension abilities would be able to correctly "understand" even benign web pages. Whether or not this happens in practice, however, is a different question.

In this paper, we focus on how cybercriminals exploit the differences between humans and detection tools in executing, parsing, and interpreting a web page. We use two signals in our investigation. The first technique we detect is textual *content obfuscation*, with which web authors prevent non-JavaScript-aware analyzers from retrieving portions of the textual content of the page, and present a different message to humans. The second technique we detect is the presence of *fake security seal* images: cybercriminals place these seals on their websites in an attempt to deceive humans (i.e., by purporting their online rouge pharmacy is "certified" by a reputable authority), even if a program would never be fooled by this practice.

We study how these techniques are used in a dataset of 149,700 web pages, containing both benign and malicious pages. Interestingly, we found that benign pages also make use of content obfuscation for specific purposes, such as making the harvesting of e-mail addresses more difficult. However, with a few heuristics (and a clustering step to eliminate outliers) we can find malicious pages with 94% precision and 95% recall among the samples that triggered this signal. The fake seal heuristic, having almost no false positives, found 400 rogue pharmacy websites.

As a proof-of-concept of how these anti-deception heuristics could be a valuable addition to many security products, we built a "maliciousness detector" leveraging signatures extracted exclusively from pages detected by our two heuristics, using the hidden text as an additional hint. While obviously not a complete anti-fraud or anti-malware solution, our tool automatically pinpointed several scam campaigns that deceive humans without exploiting any technical vulnerability, and would therefore be out of the reach of many traditional malware detectors, unless they had specific signatures for them.

Given the importance of scam campaigns and how large exploitation campaigns were found to use content obfuscation, we estimate that our heuristics could be a valuable addition to many security products, as our proof-of-concept tool has a high precision (95%) and a good recall (around 73%) when used to find any malicious page in our dataset.

To summarize, our main contributions are:

– We introduce a novel approach to detect content obfuscation, and we study its legitimate and malicious uses in a large dataset.

– We introduce a novel approach in detecting counterfeited, or just plainly fake (with no certification authority issuing them), certification seals.
– We show that this type of heuristics can be helpful to general security tools, by introducing a similarity measure and a matching system that expand their reach.

2 Related Work

To be effective, fraud pages need to deceive twice: like any malicious site, they need to convince automated analyzers that they are legitimate sites; moreover, they need to convince humans into falling for the "phish," a trait that is unique to this cybercrime branch. To identify these sites, researchers have devised detection systems that go after both of these deceptions.

Honeyclient Evasion. Researchers have identified malware/phishing sites that perform server-side cloaking to prevent honeyclients from reaching the phishing content: only real users get to the phish, whereas honeyclients get delivered legitimate content. The cloaking may happen on the redirection chain leading to them [21], or the server hosting them [42]. Other researchers have pinpointed suspicious URLs by detecting attempts to evade honeyclients analysis, typically through fingerprinting and obfuscation [18].

Blacklist Evasion. Cybercriminals have also mitigated the efficacy of successful detections by churning through a large set of domains and URLs, with domain flux and URL fluxing [11, 26, 38, 25]. Researchers, in turn, noting that this behavior is generally associated with malicious sites, have used it as a detection feature [20, 31]. These fluxing infrastructures are also being detected mining their topology [17], the redirection chains leading to them [24, 40, 21], and the traffic distributors' system feeding them a stream of users to exploit [22].

Studies on Human Scamming. Another direction of research concentrates on why humans get scammed. Sheng et al. have proposed a game [37] that teaches people about how not to get scammed. Later, demographic studies have shown that education is effective in reducing the efficacy of scams [36], but it does not solve the problem alone. Wu et al. show that security toolbars do not help the users in their assessments [45]. In 2014, Neupane et al. [29] have taken these studies a step further, using fMRIs to analyze how the brain responds to phishing sites and browser countermeasures.

Browser Phishing Warnings. Traditional browser solutions to help users be aware of the phish, such as domain highlighting and phishing warnings, have shown to be not very effective [23, 29]. To better inform users, researchers have proposed in-browser protection systems. Spoofguard [5] verifies that user sensitive data is not passed to sites with similar sounding domain names and that contain similar images. AntiPhish [19] tracks sensitive information and informs the user whenever those are given to any untrusted website. DOMAntiPhish [33] alerts the user whenever she visits a phishing site with a layout similar to a trusted website. All these solutions help in preventing that the user is deceived in trusting a site similar to a known site she used in the past, but they do not prevent against

other categories of scams, such as fake pharmacies and rogue antiviruses [7, 39]. In contrast, our system is able to track advanced, previously unseen phishing attacks.

Content Analysis. The idea of extracting text from images through OCR has been investigated in the context of e-mail spam [10]. These scams use "salting" tricks to confuse analyzers while still getting the right message to humans [4]. To counter this, researchers have proposed ways to track concept drift [8], which spammers use to thwart frequency-based content analysis. Comprehensive studies on content analysis have been proposed both for spam [30] and phishing sites [46, 50]. Google is also performing phishing detection through content analysis [44], and researchers have used the search engine's index to identify scams campaigns with similar content [16]. In contrast, our system aims to identify the advanced phishing attacks that evade these content-based solutions, obfuscating their content to be resilient to static analyzers.

Visual Analysis. Phishing pages have been identified through image shingling [1], which involves fragmenting screenshots of the phishing sites into small images, and clustering the sites according to the fraction of identical images. This solution is attractive because it is resilient to small changes in the phishing site, as long as the overall template is not altered. Previous solutions involve clustering according to colors, fonts, and layout [27] to identify phishing sites visually similar to trusted sites. Hara et al. [15] show that, given a large enough dataset of phishing sites, it is possible to automatically infer the site they are mimicking. These solutions are effective as long as the phishing attack is trying to mimic the aspect of a trusted website, but they do not cover other scam categories, such as fake pharmacies, dubious online retailers, or rogue antiviruses. In contrast, our approach uses visual analysis to identify a dead giveaway of a scam: a fake security seal. Scammers use these seals to claim to be a legitimate business, even though the company generating these security seals has not assessed the scammers' business (and often does not even exist). Focusing on these seals, we can identify scamming sites that do not try to mimic a legitimate site, but are still effective in deceiving the user.

3 Dataset

Throughout the paper, we will refer to a dataset comprising the home pages of the 81,000 most popular websites according to the Alexa popularity ranking, as a baseline of benign pages. We obtained the remaining 68,000 pages from the Wepawet online analyzer [43, 6].

Wepawet receives submissions from a variety of sources, including a large volume of URLs from automated feeds, both benign and malicious. As such, it represents a reasonable sample of pages that a security tool would be called to examine in practice. Notice that we used Wepawet merely as a feed of active URLs, without considering the results of its analysis.

In particular, we obtained two feeds: The first was an archive of 18,700 pages pre-filtered (via a simple keyword search in the URL) to contain a large number

of fake antivirus (fake AV, [39, 32]) pages, so that we could test our heuristics against this type of scam, regardless of Wepawet's ability to detect it. The second feed consisted of 50,000 submitted URLs, received in real-time so we could immediately perform our analysis on them.

Table 1. Reference truth data for the 50,000 received submissions

Page type	Samples	Percent
Pharmacy scam campaigns	2431 ± 215	$4.87 \pm 0.43\%$
"Blackhole" exploit kit	140 ± 52	$0.28 \pm 0.10\%$
Updated version of the "Blackhole" exploit kit	47 ± 29	$0.09 \pm 0.06\%$
Fake video codec scam campaign	327 ± 80	$0.65 \pm 0.16\%$
Other pages with questionable content	196 ± 62	$0.39 \pm 0.12\%$
Total number of malicious or questionable samples	3075 ± 239	$6.16 \pm 0.48\%$
Total number of benign samples	46834 ± 241	$93.79 \pm 0.48\%$

This last subset will be the basis of our final evaluation, and, as expected for a random selection, it contains a number of common scams, exploit kits, and a majority (around 94%) of benign samples. We obtained truth data for this feed through a manual review: Table 1 details our findings. For obvious time reasons, we could not examine all 50,000 pages: we opted instead for a random sample of 3,000 from which we extrapolate the totals on the entire set within a reasonable margin (Wilson confidence intervals, 85% confidence). Whenever we present these numbers, we will express them as $x \pm m$, indicating the interval $(x - m, x + m)$.

We encountered several samples with suspicious characteristics, but for which a clear benign-malicious verdict would have required a full investigation of the service provided (i.e., a "proxy service," found at several URLs under different names, and without any stated policy or identification). We marked these samples as "questionable" and have excluded them from benign and malicious sample counts.

4 Content Obfuscation

As mentioned in the introduction, many automated systems parse web pages: Examples include organizations performing large-scale crawling of the Internet or online analyzers that must evaluate the benign or malicious nature of a page within a certain time limit. These systems should view and "interpret" pages exactly like a human would, but in practice they may have to compromise accuracy in order to save bandwidth (e.g., by not downloading images) or processing time (e.g., by ignoring JavaScript, not building the actual layout of the page, not applying style sheets, etc.).

In general, content can be considered obfuscated if it would be easily seen and interpreted by the average computer user, but would be hard to interpret without simulating a full browser and the interaction of a human with it.

To refine this definition, we need to consider how automated crawlers parse pages. Details can vary a lot, but we can pick a meaningful upper bound on automated extraction capabilities and use it to differentiate the two views of a page. In particular, we conjecture that JavaScript will be a problematic feature for programs to consider, and one that many will sacrifice.

Our intuition is motivated by the consideration that analyzing a static page can be assumed to take a time that is roughly a function of its size, while executing arbitrary code introduces an unpredictable delay in the analysis. Moreover, since JavaScript code can interact with other elements on the page, the entire content must be kept in memory during the analysis (as opposed to discarding content that has already been consumed).

It is impossible to directly gauge how many web analyzers and crawlers avoid JavaScript, although, as mentioned, many search engines do provide guidelines for webmasters that are consistent with the necessity to expose content to spiders that ignore JavaScript [14, 28, 47] and images [12, 28, 48], and many published detection systems use static signature matching (e.g., [30, 35, 42]). The choice of this boundary between the "human-browser" world and the "automated-parser" world has also been indirectly confirmed by our findings presented in Section 4.3: Many pages (both benign and malicious) use JavaScript code to hide information they do not want exposed to automated parsers.

4.1 Heuristic

Several encapsulation and encoding schemes are used to transfer text on the web. A banking web site might, for example, provide content encoded in UTF-8, compressed by the server with *gzip* and transferred over a TLS connection. From the point of view of the client, however, these schemes are completely transparent: once the encoding layers are removed, a well-defined payload is reached for every data transfer.

A key observation is that text will almost always appear as-is in the payload.

In the simplest case an HTML page will be retrieved from the network, its content will be parsed in a tree, a layout will be constructed, and the resulting page will be presented to the user. The browser will simply copy the content of text nodes from the original payload, HTML entities being the only exception to this rule.

Web pages can also use scripts to dynamically add content to the page tree. This content may have already been present in the original payload (as a JavaScript string literal, for example), or it may come from additional network requests. This text does not have to be sent as-is: the script that loads the text is free to mangle and re-code it as it sees fit. There is, however, little reason to do so, and it seems safe to assume that legitimate websites will never engage in such practices: In fact, by comparing the dynamically constructed page with the

observed payloads, we have been able to confirm that textual data is transferred as-is in the vast majority of cases.

There are a few fundamental reasons for this. For purely textual content (at the sizes typically seen in web payloads), the built-in gzip compression is better, needs no extra code transfers, and is far easier to use than custom code; in fact, none of the JavaScript code compression tools in popular use significantly alter string literals. Popular data transfer formats such as JSON and XML are also text-based. There is also a historical preference for human-readable data on the web (the HTTP protocol itself is an example), both to help debugging and to ease interoperability.

While we have observed some overly-cautious escaping of content, most cases where text was not transferred as-is were due to deliberate obfuscation attempts.

One can certainly devise obfuscation systems that necessarily require human interaction and would hide text from this and any other automated detection attempt: For example, the sensitive content can be encrypted and the password presented in an image that cannot be easily parsed automatically (using, for example, the same obfuscation techniques used for CAPTCHA test images). However, this involves asking victims to perform a task that is difficult to automate. Therefore, these techniques necessarily create significant inconvenience for the intended targets, who may not have a particularly strong motivation to interact with an almost-empty page (content shown before the verification is also available to automated analyzers) if it requires effort on their part. These kinds of expedients would also strongly differentiate the page from regular benign sites, even in the eye of an untrained user, so they are unlikely to be used in fraudulent pages and were not observed in our dataset.

4.2 Implementation

Based on the discussion above, we detect content obfuscation by first building an extractor that page authors expect to face (e.g., a static text parser) and a more powerful one (e.g., a full browser), and then observing the differences in the extracted contents.

Our detector is based on the popular Firefox web browser, modified in order to observe and record all network requests and the context in which they were made. We also store the DOM tree of the page (including frames) and take a screenshot of the website as it would be seen by a human user. Finally, the browser has been modified to automatically confirm all file save request and dismiss all JavaScript popups. While using a real web browser may be slower and less safe than using an ad-hoc solution, it also makes the simulation much more realistic and helps us avoid being fingerprinted by an attacker. The browser visits the page in a temporary VM, and is fully automated.

The system uses two viewpoints to check for text on the web page. First, the text present in the DOM tree is normalized by replacing HTML entities and URL-encoded characters, removing non-alphabetic characters, and transforming all text to lowercase. Then, the text that is present in network payloads is

parsed and decoded using information recorded from the browser (for example, unzipped), and then normalized in the same way.

For every word in the body of the page (text of the DOM tree), an origin is sought in the network payloads. In particular, the presence of a word is considered "justified" if it satisfies one of the following conditions:

1. It appears as-is in the body of one of the responses from the server, including AJAX requests, requests made from iframes, etc. This rule matches most regular text.
2. It appears in one of the URLs (e.g., as part of the domain name, in the query string, etc). This rule takes care of pages that display information that is passed to them via the URL and access it via the `location` object.
3. It appears in one of the HTTP headers, which are readable by JavaScript for AJAX requests.
4. It can be obtained from the previously-mentioned sources:
   (a) as the concatenation of two words,
   (b) as the truncation of another word.
5. It is found in a whitelist of words that can be obtained directly from the JavaScript language interpreter, including type names like *none* or *HTMLParagraphElement*, date components (e.g., month names), strings from the navigator object, etc.

These rules attempt to construct a set of benign clear-text sources that would be easily exposed to a static signature matcher, anticipate most real-world string operations (that any signature matching algorithm can and probably should take into account), and consider all page components that would possibly be exposed. Words for which an origin could not be pinpointed are considered obfuscated. For simplicity, our heuristic operates on single words only, and leaves the extension to sequences to the signature generator (Section 6.1). Purely image-based obfuscation approaches would also escape our textual detector and would require image-processing techniques for detection: similarly, we left this extension to our proof-of-concept signature matcher (Section 6.2).

The previously-described extraction rules are also applied to the domain names of URLs present in HTML attributes. This allows us to catch cases of pages that try to hide from programs the URL to which they will redirect a human viewer. Notice that text from a DOM tree is analyzed considering only network payloads that have been seen before it was retrieved, so the presence of links in a page cannot be justified with future HTTP headers.

In principle, our algorithm would also work for scripts and style sheets, but such analysis is out of scope for this paper: ordinary human users do not "parse" them, nor are they aware of their existence.[1]

[1] Of course, a possible extension of our work would be to consider the two "views" of a malware analyst and of a web browser. As an example, in this model JavaScript obfuscation would be a case of content easily interpreted by a browser but cumbersome for a human to understand.

4.3 Evaluation of the Detection of Obfuscated Content

Table 2 presents the samples for which we found obfuscated content (the benign, malicious, or questionable nature was established by manual review).

While obviously not perfect, our heuristic presented very few false positives in the detection of obfuscated content (that is, content incorrectly marked as obfuscated, regardless of the benign or malicious nature of the page) in our dataset. These were often caused by incorrect parsing of the network payload, as in some cases it is difficult to replicate the exact parsing the browser will perform: unclear specifications, buggy servers, and sloppy page coding practices require browsers to rely on several heuristics, and previous research has shown that different clients can even give different interpretations to the same data [2]; it should be noted, however, that the desire to increase the number of victims can be a mitigation factor for this issue, especially for frauds: their authors have an interest in having them functional on all major browsers.

Table 2. Samples found by the content obfuscation heuristic, grouped by source

Source	Page type	Samples	In-Feed Pct.	Global Pct.
Alexa ranking (81,000 samples)	Benign	52	90%	0.06%
	Questionable	3	5.2%	0.004%
	Malicious	3	5.2%	0.004%
	total	*58*	*100%*	*0.07%*
Fake AV feed (18,700 samples)	Benign	1	0.93%	0.005%
	Questionable	4	3.7%	0.02%
	Malicious	102	95%	0.54%
	total	*107*	*100%*	*0.57%*
Received submissions (50,000 samples)	Benign	3	3.1%	0.006%
	Questionable	0	0%	0%
	Malicious	94	97%	0.19%
	total	*97*	*100%*	*0.2%*
All feeds (149,700 samples)	Benign	56	21%	0.037%
	Questionable	7	2.7%	0.005%
	Malicious	199	76%	0.13%
	total	*262*	*100%*	*0.18%*

Manual review of the 3,000 randomly selected samples from the 50,000 submissions received in real-time (Section 3), utilizing the screenshots and assisted by Optical Character Recognition software, did not uncover any false negative (obfuscated content not marked as such).

4.4 Observed Uses of Obfuscation

Even a simple `unescape` is enough to hide content from a straightforward HTML parser, and many examples in our dataset did not go much further than that. Code from exploit kits and fake antivirus scams, on the other hand, went to great lengths to obfuscate both the content and the generating script.

Beside fraudulent content, the following categories of text were commonly observed in obfuscated form:

- **E-mail addresses:** A precaution against address-harvesting spam bots. To avoid these false positives, our heuristic ignores all `mailto:` links and strings that look like e-mail addresses.
- **Domain names:** A pharmacy scam campaign and several exploit kits presented landing pages with redirection code. The target URL was obfuscated, presumably to slow down blacklist-building crawlers and human analysts.
- **Links:** Some benign websites obfuscated hyperlinks to other pages or websites, including seemingly innocuous links such as those to contact information and the terms of service. This is probably done to make sure that search engines do not focus on pages that are perceived as not significant by the webmaster; this particular technique may be an answer to Google's de-emphasizing of the `nofollow` attribute [9].

Our detector does not mark pages as questionably obfuscated if e-mail and links are the sole hidden content types, as these kinds of obfuscation are also used in benign sites.

Obfuscated target URLs, on the other hand, are definitely a strong signal of malicious content. However, the specific URLs are highly variable, easily changed, and not necessarily exposed to the user (who will likely only view and act on the first URL in the redirect chain), so they have also been ignored for the purposes of our proof-of-concept detector.

4.5 From Obfuscation Detection to Maliciousness Detection

As mentioned, our simple heuristic is, in itself, an indication that a page may contain suspicious content, but is not entirely reliable as a maliciousness detector in itself.

It does, however, find content that the page author wanted to hide: a good starting point for a more punctual detection of maliciousness. In our study, we chose to exploit the fact that cybercriminals typically run campaigns (they typically prepare many variations that implement a certain scheme) or implement general fraud schemes (such as fake pharmacies or fake antiviruses), whereas benign usages are much more likely to appear as outliers.

To this end, we leveraged obfuscated words as features for each page. Several methods exist to eliminate outliers: we opted for density-based clustering (DBSCAN), as it performs well and can be fully automated for this purpose as long as known-benign obfuscated samples are available.[2]

[2] Specifically, we rely on the presence of a few samples originated from the Alexa set for our purposes: Intuitively we want the clustering to consider them as "noise," so (for the purpose of this step) we classify them as "benign" and everything else as "malicious". With this assignment, we have a rough estimate of how good each possible clustering is (using, for instance, the F_1 score) at discriminating between benign and malicious samples. At this point, a simple grid search can find the values for the two DBSCAN parameters (the point neighborhood size ε and the minimum cluster size) that maximize this estimated score. On Table 2's data, 0.82 and 3 were found by the grid search; the corresponding clustering is shown in Table 3.

Table 3. Samples found by the content obfuscation heuristic, automatically grouped by cluster and de-noised leveraging the obfuscated words that were detected

	Page type	**Samples**
Cluster 1	"Blackhole" exploit kit	41
Cluster 2	First fake-AV campaign	23
Cluster 3	"Blackhole" exploit kit	20
Cluster 4	First fake-AV campaign (minor variation)	16
Cluster 5	Second fake-AV campaign	5
Cluster 6	Third fake-AV campaign	9
Cluster 7	Updated version of the "Blackhole" exploit kit	12
Cluster 8	False positives	4
Cluster 9	Fake flash player campaign	4
Cluster 10	"Blackhole" exploit kit	12
Cluster 11	False positives	3
Cluster 12	Third fake-AV campaign (minor variation)	15
Cluster 13	Third fake-AV campaign (minor variation)	20
Cluster 14	Third fake-AV campaign (minor variation)	4
Cluster 15	Updated version of the "Blackhole" exploit kit	8
Cluster 16	False positives	3
Cluster 17	False positives	3
Samples discarded as noise	7 questionable, 3 fake-AV campaigns (all minor variations of the campaigns above), 3 pharmacy scams, 1 "Blackhole" exploit kit (bugged sample), 3 "get rich quick" scams, and 43 benign	60

This step is not strictly necessary, as far as finding interesting and suspicious pages: Without further refinement, the heuristic already pointed to 199 truly malicious pages (and 56 benign ones), many of which were not originally found by Wepawet's analysis, especially among the scam campaigns that did not leverage any browser vulnerability.[3]

However, we found this step very useful both for our manual analysis and for a more punctual malicious content detection. Its results are presented in Table 3, which also serves as a recap of the nature of the pages found by the heuristic of this section. Confirming the validity of our intuition that most benign pages would appear as outliers, this clustering step was able to achieve a precision of 93.56% and a recall 94.97% in finding malicious pages among the samples that presented obfuscation.

The false positives are multiple benign sites that were obfuscating a few similar words, usually for search engine optimization purposes. All fake antivirus

[3] Interestingly, while reviewing the pages that were found, we even encountered several that appeared to be generated by an exploit kit, although Wepawet had not detected them as such. Further review revealed that these pages, generated by the "Blackhole" exploit kit, were fingerprinting Wepawet's JavaScript engine and disabling their malicious payload to escape detection.

campaigns present in the dataset were identified correctly. As mentioned, pages from two well-known exploit kits were also identified.

As expected, campaigns tend to emerge as distinct clusters. Incidentally, we have observed a certain number of minor variations, updates, or bugs within the same campaign or usage of exploit kits: this tends to surface as a splitting of a campaign in a few distinct clusters.

5 Counterfeit Certification Seals

Our second heuristic explores an attempt to confuse human consumers without attempting to deceive automated analyzers.

In particular, we observed that in many cases scammers try to make their pages appear more legitimate by including certification seals: small images meant to convey that the site has passed a review by a trusted third party. These seals are also often displayed by legitimate online sellers to reassure users about the safety of their data. Reputable companies releasing these certifications include Symantec, GeoTrust, McAfee and other well-known certification authorities and vendors of security software.

No standard mandates the exact meaning of the certification. Some issuers just claim to periodically check the website for malware, others are meant to fully verify that the site is owned by a reputable business entity. In all cases, seals are included to make visitors more comfortable (and presumably more likely to spend time or money on the site). As such, their counterfeiting is attractive for fraudsters, even if no computer program would "understand" them or consider them in any way.

5.1 Use by Fraudsters

Unfortunately, there are also no standards on how certification seals should be included in a page and how an end user can verify their legitimacy. Unlike HTTPS certificates, browsers cannot check them on the user's behalf, as the seal is usually just another image on the page.

Issuers can mandate certain technical measures in their usage policies, such as the requirement to include a script served from an authority's server [41]. Typically, correctly included seals should react to clicks by opening a verification page hosted by the issuer.

Nothing, however, prevents a malicious seller from simply copying the seal image from a legitimate site and displaying it on a fraudulent page. Should someone click on the seal to verify it, the scammer can simply present a locally-hosted fake certification. Unless the end user specifically checks the certification page origin (and knows the correct domain name of the authority that issues the seal in question), the page will look legitimate.

Given the ease of including a copied image and the low risk of detection by untrained users, fraud perpetrators often display copious amounts of certification

Fig. 1. Examples of counterfeit certification seals found on rogue pharmacy sites.

seals on their pages, especially on online shops such as rogue pharmacies. Figure 1 shows a few examples.

Seal images can be completely made-up and refer to no established third-party, present alterations of logos of real certification authorities [3] or, as we most commonly observed, present copies of actual logos and faked certification pages.

5.2 Heuristic

For our study, we augmented the system described in Section 4.2 with a component that calculates a perceptual hash [49] (for resilience against small alterations) of all images with a size comparable to the ones typically used for certification seals, compares them with the known ones, and checks if they are legitimate or not.

There are few legitimate seal providers: a manual review of their terms of services and inclusion practices would allow constructing a fully reliable detector, if desired. As expected for a deception technique exclusively directed toward humans, we did not observe any attempt to hide its use from even a simple analyzer. Therefore, we opted again for a fully-automated approach in our survey: we performed optical character recognition on the 100 most common images (as aggregated by the perceptual hashing function) and looked for keywords expressing trust and protection such as "secured," "approved," "trust," and "license" to find seals, and check legitimacy simply by verifying if they link off-site or not: an imperfect approach that however highlights how easy it can be for a program to detect purely human-directed deception attempts.

While not uncovering all frauds in our dataset (not all of them use these fake seals, nor does our heuristic cover all of these images), this simple heuristic correctly flagged about 400 samples, with no false positives. All these samples originated from rogue pharmacy campaigns and, as we will show in the next section, proved to be a valuable starting point for a detector of this entire category of scams.

6 Proof-of-concept General Detector

As we have seen, the difference between a human view and an algorithmic view can indeed be useful in pointing out malicious pages, even with two simple

heuristics such as ours. Of particular note is its tendency to find "pure" scam campaigns that do not involve software exploits, yet succeed due to deception.

In this section we will show if these heuristics could be useful to a complete maliciousness detection suite, in particular by seeing if the pages they uncover could enable finding more. To this end, we implemented a proof-of-concept detector that uses them as its only starting point, and we will show how it can already reach significant detection rates. It is fairly standard in its construction (signature generation and matching), but we will also use it to exemplify a similarity measure that gives a different "weight" to certain words (the ones that were obfuscated, in our case) and is resilient to the inclusion of extraneous text, as we have observed this happens with a certain frequency in scam pages when they are part of a larger page (posts in hijacked forums are an example). Incidentally, this approach would also defend against fraudsters including large amounts of irrelevant text specifically to thwart automated analysis, even if it was presented in such a way that humans would not pay attention to it (i.e., in a semi-invisible color, at the end of a long page, out of view, ...).

6.1 Signature Generation

The clusters in Table 3, with the addition of the pages detected due to seal counterfeiting, serve as the basis to generate signatures. In particular, our system tries to identify contiguous regions of text that are "typical" of a cluster, to maximize the impact of common textual elements (presumably core to the nature of those pages), while de-emphasizing regions that are variable among the different samples: A score is assigned to each word present in pages belonging to the cluster. The score is initially the number of occurrences of the word in the samples, doubled if the word was obfuscated; scores are then normalized to have zero-average (to further reduce noise, we also exclude the 100 most common English words). All the maximal-scoring contiguous regions of text are then found (this operation has linear-time complexity [34]) and identical regions are aggregated to form the "signature" regions for that cluster.

As an example, for the cluster of a simple fake-AV campaign that included a few variations, the following regions were chosen: `center initializing virus protection system`, `initializing virus protection system`, `initializing treat protection system.`, whereas for a pharmacy scam regions included both the entire common content of the typical sales page, and smaller text snippets that were present in many, but not all pages (mainly, the type of drugs sold in specific subpages).

6.2 Signature Matching

When presented with a sample, our proof-of-concept detector will perform a fuzzy matching with the signature regions, to find other similar but unknown campaigns.

Simple textual similarity measures (i.e., the Jaccard coefficient) weight the amount of common elements versus the amount of uncommon ones. As mentioned,

we will propose here a slightly different approach that is more resilient to the inclusion of unrelated random words in the page, as we consider this a good property when faced with content that exclusively tries to deceive humans: in those cases, a small amount of information could very well be sufficient.

In particular, when evaluating a page:

1. A candidate match m_i is found for each of the n signature regions s_i (longest matching subsequence in the page text).[4]
2. The Jaccard distance d_i is computed for each (m_i, s_i) pair.
3. The distance of the page to the clusters is computed: $d = \min\{d_1 \dots d_n\}$.

Notice that with this method only one zone of the page influences the final result: the one that is most similar to one of the clusters. Therefore, as opposed to inserting disturbances anywhere, an author that wished to avoid detection would have to modify several words right in the middle of their most relevant content, likely changing the message perceived by a potential victim.

At this point we can mark a page as benign or malicious based, using a threshold on d. To make sure the threshold is neither too low nor too high, we used a separate training phase to select a good value.[5]

To exemplify extra robustness precautions that would be included in a complete detector, we added two image-based matching systems based on a page screenshot. One recovered text through Optical Character Recognition (which can then be used as normal HTML text), the other directly compared the page screenshot with those of the pages found by the two heuristics.

6.3 Evaluation

We evaluate the overall performance of our proof-of-concept system on the (otherwise unlabeled) 50,000 samples obtained from real-time submissions. As mentioned in Section3, this set includes a variety of scam sites, traditional drive-by download exploit pages, and benign pages.

Table 4 provides a numerical overview of its performance. Overall, the system flagged 1,833 pages as malicious. Based on manual analysis of these instances, we confirmed 1,746 cases as true positives (first line): a significant increase from the

[4] If desired, each match m_i could also be enlarged by a percentage to achieve extra resiliency against the insertion of random "stopping" words in the middle of an otherwise matching page section.

[5] We used a procedure similar, in its principle, to the one employed in Section 4.5: Pages from the Alexa feed were all marked as benign, then a well-scoring threshold value was computed for each cluster. Again, those known-benign pages are used to get a rough estimate of the amount of false positives that each given value would cause. A linear search is then used, starting from a low threshold value and increasing it until the false positive rate surpasses a certain percentage: 2%, in our case. We protect from possible outliers in the Alexa feed by requiring the presence of at least three of its samples before the search is terminated. While this procedure requires a sizable number of benign samples, it is applicable in many cases: such samples are easily gathered from widely-available rankings or website directories.

95 detected by the content obfuscation heuristic (Table 2) and the 400 detected by the seal-counterfeiting one; the remaining 87 flagged pages were incorrect detections of benign pages (second line). As anticipated in Section 3, negatives are expressed as confidence intervals (lines three and four, and dependent scoring values in the remaining lines).

Table 4. Performance of the proof-of-concept detector

True positives (flagged, malicious)	$1,746$
False positives (flagged, benign)	87
True negatives (not flagged, benign)	$47,524 \pm 192$
False negatives (not flagged, malicious)	643 ± 192
Precision	95.25%
Recall	$73.08 \pm 5.86\%$
F_1 *score*	$82.69 \pm 3.75\%$
True positive rate	$73.08 \pm 5.86\%$
False positives rate	$0.18 \pm 0.00\%$

True positives included pharmacy scams from several different campaigns or sellers and many fake-AV variants, underlining how the findings from these heuristics generalize well and improve detection (most scam campaigns, for instance, were not originally marked as suspicious or malicious by Wepawet). Examples of false negatives are posts on hijacked forums that contained relatively little malicious content, or scams that significantly differed from the samples found by our two simple heuristics. For cases where the samples were related (although quite different and possibly originating from different criminal operations), our system showed excellent detection.

While not enough to create a complete security system, these results confirm our intuition that detection of this "view difference" (even in the two simple heuristic forms we presented) can be a useful addition to many analyzers. As an example, referring to Table 1, a honeyclient or exploit-based detector is unlikely to catch the (otherwise very common) scam campaigns without a method such as ours.

Finally, we note that even just removing content obfuscation and counterfeit certifications from the fraudsters' tool arsenal could be a desirable result in itself. In fact, a general property of heuristic like ours is that they present attackers with a problematic choice: from their point of view, if they choose to exploit the difference between the two "worlds" of programs and humans they risk giving away the nature of their operation (and possibly uncover other similar scams). Presenting the same information to humans and programs, on the other hand, will make it easy for security researchers to construct reliable signatures for the malicious campaign and quickly reduce its impact.

7 Conclusions

In this paper we pointed out how the discrepancy between the understanding of a human and a program can present both a danger (as a way for cybercriminals to escape analysis) and a maliciousness-detection opportunity at the same time. We presented two heuristics that detect cases that exemplify this situation: one directed toward textual content and one involving images, respectively countering the deception of static analyzers and of human beings.

Envisioning that these detection methods can complement existing detection tools, we have implemented a proof-of-concept detector based exclusively on these methods to discover online malicious pages. This tool alone was able to achieve a 95% precision and a 73% recall in our dataset, and was able to discover a high number of human-directed fraud pages that would otherwise be outside the detection capabilities of most traditional malware analyzers.

Acknowledgments. We would like to thank Davide Paltrinieri for his help and ABBYY for providing the OCR software.

This work was supported by the Office of Naval Research (ONR) under Grant N000140911042, the Army Research Office (ARO) under grant W911NF0910553, and Secure Business Austria. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the Office of Naval Research, the Army Research Office, or Secure Business Austria.

References

1. Anderson, D.S., Fleizach, C., Savage, S., Voelker, G.M.: Spamscatter: Characterizing Internet Scam Hosting Infrastructure. In: Proceedings of the USENIX Security Symposium (2007)
2. Barth, A., Caballero, J., Song, D.: Secure Content Sniffing for Web Browsers, or How to Stop Papers from Reviewing Themselves. In: Proceedings of the 30th IEEE Symposium on Security and Privacy. IEEE (2009)
3. Bate, R., Jin, G., Mathur, A.: In Whom We Trust: The Role of Certification Agencies in Online Drug Markets. NBER working paper 17955 (2012)
4. Bergholz, A., Paass, G., Reichartz, F., Strobel, S., Moens, M.F., Witten, B.: Detecting Known and New Salting Tricks in Unwanted Emails. In: Proceedings of the Conference on Email and Anti-Spam (CEAS) (2008)
5. Chou, N., Ledesma, R., Teraguchi, Y., Mitchell, J.C.: Client-side Defense Against Web-Based Identity Theft. In: Proceedings of the Network and Distributed System Security Symposium (NDSS) (2004)
6. Cova, M., Kruegel, C., Vigna, G.: Detection and Analysis of Drive-by-download Attacks and Malicious JavaScript code. In: Proceedings of the World Wide Web Conference (WWW) (2010)
7. Cova, M., Leita, C., Thonnard, O., Keromytis, A.D., Dacier, M.: An Analysis of Rogue AV Campaigns. In: Proceedings of the International Symposium on Research in Attacks, Intrusions and Defenses (RAID). pp. 442–463 (2010)

8. Cunningham, P., Nowlan, N., Delany, S.J., Haahr, M.: A Case-Based Approach to Spam Filtering that Can Track Concept Drift. Knowledge-Based Systems (2005)
9. Cutts, M.: Pagerank sculpting. http://www.mattcutts.com/blog/pagerank-sculpting/ (2009)
10. Fumera, G., Pillai, I., Roli, F.: Spam Filtering Based on the Analysis of Text Information Embedded into Images. The Journal of Machine Learning Research 7, 2699–2720 (2006)
11. Garera, S., Provos, N., Chew, M., Rubin, A.D.: A Framework for Detection and Measurement of Phishing Attacks. In: Proceedings of the ACM Workshop on Recurring Malcode (WORM) (2007)
12. Google Inc.: Image publishing guidelines. http://support.google.com/webmasters/bin/answer.py?hl=en&answer=114016 (2012)
13. Google Inc.: Making AJAX Applications Crawable. https://developers.google.com/webmasters/ajax-crawling/ (2014)
14. Google Inc.: Webmaster Guidelines. http://support.google.com/webmasters/bin/answer.py?hl=en&answer=35769#2 (2014)
15. Hara, M., Yamada, A., Miyake, Y.: Visual Similarity-Based Phishing Detection without Victim Site Information. In: Proceedings of the IEEE Symposium on Computational Intelligence in Cyber Security (CICS). pp. 30–36. IEEE (Mar 2009)
16. Invernizzi, L., Benvenuti, S., Comparetti, P.M., Cova, M., Kruegel, C., Vigna, G.: EVILSEED: A Guided Approach to Finding Malicious Web Pages. In: Proceedings of the IEEE Symposium on Security and Privacy (S&P) (2012)
17. Invernizzi, L., Miskovic, S., Torres, R., Saha, S., Lee, S.J., Mellia, M., Kruegel, C., Vigna, G.: Nazca: Detecting Malware Distribution in Large-Scale Networks. In: Proceedings of the Network and Distributed System Security Symposium (NDSS) (2014)
18. Kapravelos, A., Shoshitaishvili, Y., Cova, M., Kruegel, C., Vigna, G.: Revolver: An Automated Approach to the Detection of Evasive Web-based Malware. In: Proceedings of the USENIX Security Symposium (2013)
19. Kirda, E., Kruegel, C.: Protecting Users Against Phishing Attacks with AntiPhish. In: Proceedings of the International Conference on Computer Software and Applications (COMPSAC). vol. 1, pp. 517–524. IEEE (2005)
20. Konte, M., Feamster, N., Jung, J.: Fast Flux Service Networks: Dynamics and Roles in Hosting Online Scams. Tech. rep., Georgia Institute of Technology and Intel Research (2008)
21. Lee, S., Kim, J.: WarningBird: Detecting Suspicious URLs in Twitter Stream. In: Proceedings of the Network and Distributed System Security Symposium (NDSS) (2010)
22. Li, Z., Alrwais, S., Xie, Y., Yu, F., Wang, X.: Finding the Linchpins of the Dark Web: a Study on Topologically Dedicated Hosts on Malicious Web Infrastructures. Proceedings of the IEEE Symposium on Security and Privacy (S&P) pp. 112–126 (May 2013)
23. Lin, E., Greenberg, S., Trotter, E., Ma, D., Aycock, J.: Does Domain Highlighting Help People Identify Phishing Sites? In: Proceedings of the Conference on Human Factors in Computing Systems (CHI). p. 2075. ACM Press, New York, New York, USA (2011)
24. Lu, L., Perdisci, R., Lee, W.: SURF: Detecting and Measuring Search Poisoning Categories. In: Proceedings of the ACM Conference on Computer and Communications Security (CCS) (2011)

25. Ludl, C., Mcallister, S., Kirda, E., Kruegel, C.: On the Effectiveness of Techniques to Detect Phishing Sites. In: Proceedings of the SIG SIDAR Conference on Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA). pp. 20–39 (2007)
26. Mcgrath, D.K., Gupta, M.: Behind Phishing : An Examination of Phisher Modi Operandi. In: Proceedings of the USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET) (2008)
27. Medvet, E., Kirda, E., Kruegel, C.: Visual-Similarity-Based Phishing Detection. In: Proceedings of the International Conference on Security and Privacy in Communication Networks (SecureComm). p. 1. ACM Press, New York, New York, USA (2008)
28. Microsoft Corp.: Bing Webmaster Guidelines. `http://www.bing.com/webmaster/help/webmaster-guidelines-30fba23a` (2014)
29. Neupane, A., Saxena, N., Kuruvilla, K., Georgescu, M., Kana, R.: Neural Signatures of User-Centered Security: An fMRI Study of Phishing, and Malware Warnings. In: Proceedings of the Network and Distributed System Security Symposium (NDSS). pp. 1–16 (2014)
30. Ntoulas, A., Hall, B., Najork, M., Manasse, M., Fetterly, D.: Detecting Spam Web Pages through Content Analysis. In: Proceedings of the International World Wide Web Conference (WWW). pp. 83–92 (2006)
31. Prakash, P., Kumar, M., Kompella, R.R., Gupta, M.: PhishNet: Predictive Blacklisting to Detect Phishing Attacks. In: Proceedings of the IEEE International Conference on Computer Communications (INFOCOM). pp. 1–5. IEEE (Mar 2010)
32. Rajab, M.A., Ballard, L., Marvrommatis, P., Provos, N., Zhao, X.: The Nocebo Effect on the Web: An Analysis of Fake Anti-Virus Distribution. In: Large-Scale Exploits and Emergent Threats (LEET) (2010)
33. Rosiello, A.P.E., Kirda, E., Kruegel, C., Ferrandi, F.: A Layout-Similarity-Based Approach for Detecting Phishing Pages. In: Proceedings of the International Conference on Security and Privacy in Communication Networks (SecureComm) (2007)
34. Ruzzo, W., Tompa, M.: A Linear Time Algorithm for Finding All Maximal Scoring Subsequences. In: Proceedings of the Seventh International Conference on Intelligent Systems for Molecular Biology. AAAI (1999)
35. Seifert, C., Welch, I., Komisarczuk, P.: Identification of Malicious Web Pages with Static Heuristics. In: Proceedings of the Australasian Telecommunication Networks and Applications Conference. IEEE (2008)
36. Sheng, S., Holbrook, M., Kumaraguru, P., Cranor, L., Downs, J.: Who Falls for Phish? A Demographic Analysis of Phishing Susceptibility and Effectiveness of Interventions. In: Proceedings of the Conference on Human Factors in Computing Systems (CHI). pp. 373–382 (2010)
37. Sheng, S., Magnien, B., Kumaraguru, P., Acquisti, A., Cranor, L.F., Hong, J., Nunge, E.: Anti-Phishing Phil: The Design and Evaluation of a Game That Teaches People Not to Fall for Phish. In: Proceedings of the Symposium on Usable privacy and security (SOUPS). pp. 88–99 (2007)
38. Sheng, S., Wardman, B., Warner, G., Cranor, L.F., Hong, J.: An Empirical Analysis of Phishing Blacklists. In: Proceedings of the Conference on Email and Anti-Spam (CEAS) (2009)
39. Stone-Gross, B., Abman, R., Kemmerer, R., Kruegel, C., Steigerwald, D., Vigna, G.: The Underground Economy of Fake Antivirus Software. In: Proceedings of the Workshop on Economics of Information Security (WEIS) (2011)
40. Stringhini, G., Kruegel, C., Vigna, G.: Shady Paths: Leveraging Surfing Crowds to Detect Malicious Web Pages. In: Proceedings of the ACM Conference on Computer and Communications Security (CCS) (2013)

41. Symantec: Seal License Agreement. https://www.symantec.com/content/en/us/about/media/repository/norton-secured-seal-license-agreement.pdf (2014)
42. Wang, D.Y., Savage, S., Voelker, G.M.: Cloak and Dagger: Dynamics of Web Search Cloaking. In: Proceedings of the ACM Conference on Computer and Communications Security (CCS). pp. 477–489 (2011)
43. Wepawet. http://wepawet.cs.ucsb.edu
44. Whittaker, C., Ryner, B., Nazif, M.: Large-Scale Automatic Classification of Phishing Pages. In: Proceedings of the Network and Distributed System Security Symposium (NDSS) (2010)
45. Wu, M., Miller, R.C., Garfinkel, S.L.: Do Security Toolbars Actually Prevent Phishing Attacks? In: Proceedings of the Conference on Human Factors in Computing Systems (CHI). pp. 601–610 (2006)
46. Xiang, G., Hong, J., Rose, C.P., Cranor, L.: CANTINA+: A Feature-Rich Machine Learning Framework for Detecting Phishing Web Sites. In: ACM Transactions on Information and System Security. pp. 1–28 (2011)
47. Yandex N.V.: Recommendations for webmasters - Common errors. http://help.yandex.com/webmaster/recommendations/frequent-mistakes.xml (2014)
48. Yandex N.V.: Recommendations for webmasters - Using graphic elements. http://help.yandex.com/webmaster/recommendations/using-graphics.xml (2014)
49. Zauner, C.: Implementation and Benchmarking of Perceptual Image Hash Functions. Master's thesis, Upper Austria University of Applied Sciences, Hagenberg Campus (2010)
50. Zhang, Y., Hong, J., Cranor, L.: CANTINA: a Content-Based Approach to Detecting Phishing Web Sites. In: Proceedings of the ACM Conference on Computer and Communications Security (CCS) (2007)