CASTLE: A Social Framework for Collaborative Anti-Phishing Databases

Arash Nourian, Sameer Ishtiaq, and Muthucumaru Maheswaran
Advanced Networking Research Lab
School of Computer Science
McGill University
Montreal, QC H3A 2A7, Canada
Email: {nourian, sishti, maheswar}@cs.mcgill.ca

***Abstract*—Phishing is a major problem on the Internet. The cornerstone of anti-phishing is detecting whether a given site is good or bad. Most of the approaches for anti-phishing rely on looking up centrally maintained repositories. In this paper, we present a decentralized framework called CASTLE that allows a collaborative approach for anti-phishing services. We implemented a prototype and then tested it on Planet-lab. The experiments indicate the viability of our framework.**

I. Introduction

The major goal of phishing attacks is to steal valuable personal (usually identity related) information from users. Once the attackers gain personal data such as passwords, date of birth, and bank account number, they use the information to their benefits by creating bogus identities or taking control of online accounts.

The phishing attacks are launched in many different ways [1]. The most common way is to send an email to users and persuade users to click on a forged link inside the email. When the user clicks on the forged link, the user will be sent to a site controlled by the attacker which looks like a trusted site (e.g., the user's online bank). Because the site is under the attacker's control, all information revealed by the user such as username and passwords are gained by the attacker. A vigilant user may detect the difference between the bogus site created by the attacker and the legitimate site. However, a sizeable population of the users do fall prey to these attacks [2]. For example, legitimate websites (depending on the sophistication of the institution) can have welcome messages that include the user's name from previous successful logins. Such login information is not available for the bogus site and consequently it cannot customize the welcome message.

The number of phishing attacks is increasing rapidly [3]. Because of their value, financial institutions used to be the favorite targets of the phishing attackers. Recently, this form of attack has diversified and started targeting social networking sites as well. Further, the technologies adopted by the attackers are getting more sophisticated by day. In June 2008, the longest time online for a phishing website was 30 days [4]. In the same month, Anti Phishing Working Group received about 28,151 unique phishing email reports [4]. Also the average uptime for a phishing website is 4.5 days [4].

CASTLE (Collaborative Approach using Social, Content, and Location for Evaluation) is a decentralized framework for building and maintaining databases useful for anti-phishing services. For instance, browser-based toolbars [2] lookup centrally maintained databases to determine whether a given location is safe. If a site is previously black-listed as phishing, the browser-based toolbar prevents the user from visiting it [5]. Although the centralized databases are already in place for anti-phishing, their scalability is a cause for concern. In particular, the existing approaches do not handle the following issues in a scalable manner.

- *Enrollment process:* The name space that can be phished is vast. Few repositories cannot handle complaints for the whole name space. Because phishing is usually an impersonation attack against well known Internet "brands," there is a concern of brand tarnishing. Therefore, valuable brands would want to take ownership of their namespace.
- *Decision process:* With few central repositories, administrators of those repositories should use automated tools and manual examinations to decide on phishing reports. Because a repository is likely to receive many reports and some of which is going to be completely new, the decision process is an arduous one. If the namespace is partitioned based on URL prefixes, the decisions can be made by repositories that are familiar with URLs. In CASTLE, the social factor is used to reduce namespace grabbing attacks.
- *Lookup process:* One of the advantages of central repositories is the straightforward lookup process for determining whether a URL is safe. This advantage will be lost if multiple repositories need to be checked. A distributed network of repositories need to have an efficient lookup process.

CASTLE framework provides a novel decentralized peer-to-peer approach for maintaining anti-phishing databases for the Internet. It uses the social factor to limit the membership to the peer-to-peer network. This is necessary to prevent the attackers from locating their verification servers in the network of servers maintained by CASTLE. Further, CASTLE admits location-based (URL-based) and content-based access to the repositories.

Digital Object Identifier: 10.4108/ICST.COLLABORATECOM2009.8310
http://dx.doi.org/10.4108/ICST.COLLABORATECOM2009.8310

II. Background

Phishing is sometimes confused with spam; however it is not spam. Spam's real purpose is to sell, while phishing's purpose is to steal. There are many different opportunities for creating phishing URLs. Following are some examples of techniques used for phishing on the Internet [6]:

- Adding a suffix to the domain name of a URL. For example, `www.citybank.com` can be changed to `www.citybank.us.com`
- Actual link is different from the visible link. For example, the HTML line: `<ahref="http://www.citibank.com.us.ebanking.us"> www.citibank.com</a>` where visible link is `http://www.citibank.com` but the actual link is `http://www.citibank.com.us.ebanking.us`.
- Using a bug in a real webpage to redirect to phishing web page. For example, the bug of eBay website: `http://cgi.ebay.com/ws/eBayISAPI.dll?MfcISAPICommand=RedirectToDomain\&DomainUrl=PHISHINGLINK` can direct you to any specified PHISHINGLINK.
- Replacing similar characters in the real link. For example replace Is (uppercase i) with l (lowercase of L) or 1 (Arabic number one), such as `WWW.CITIBANK.COM` to `WWW.C1TlBANK.COM`.
- Encoding the URL to disguise its true value using hex, dword, or octal encoding. For example, `http://www.visa.com@%32%32%30%2E%36%38%2E%32%31%34%2E%32%31%33` translates into 220.68.214.213 instead of `www.visa.com`.
- The URL is actually a part of an image, which uses map coordinates to define the click area and the real URL, with the fake URL from the <a> tag being displayed.
- Using a URL masking service such as `cjb.net` or `tinyurl.com`. For example, `http://jne9rrfj4.CjB.neT/?uudzQYRgY1GNEn` can hide actual URL and redirect the user to the phishing website.
- Link points to a page on a legitimate site which can redirect the user to the phishing website. For example, `http://www.google.com/url?q=http://www.geocities.com/mibmib4321/` where the mibmib4321 site contains a redirect to 218.214.130.51.
- Using the "@" sign. Everything to the left of "@" is ignored while everything to the right is considered. For example, `http://www.usbank.com/update.pl@81.109.43.102/usb/upd.pl` redirects user to `81.109.43.102/usb/upd.pl` instead of `http://www.usbank.com/update.pl`.
- Using a credible looking text string within the URL. For example, `http://81.109.43.102/ebay/account_update/now.php` uses ebay keyword in the URL.

Once the user is lured to the bogus website, the attacker needs to present a convincing user experience. The techniques used by the attackers to create bogus web sites can be categorized in three different ways [6]:

- Using the downloaded webpage from real website to make the phishing webpage appear and react exactly the same as the real one.
- Using script or add-in to web browser to hide the address bar in order to spoof users to believe they have entered the correct website.
- Using visual based content (E.g., image, flash, video, etc.) rather than HTML to avoid HTML based phishing detection.

III. Related Work

In recent years, a number of anti-phishing techniques have been developed and in this section, we highlight a subset of those anti phishing techniques, which are closely related to our work.

EBay toolbar [7] can identify if any particular URL which is trying to phish `ebay.com`. However, the problem is that this toolbar does not work if a URL is trying to phish some other website.

Microsoft and AOL used *Black List Approach* [8] [9] by integrating black list-based anti phishing support into their browsers. It blocks users from entering any information while he/she is at a known phishing website. However it is ineffective because there is always a window of vulnerability during which users are susceptible to attacks.

PhishingGuard [10] makes use of a white listing approach. The basic idea is that a website is identified by its IP address. The white list contains only trusted URLs. Whenever any website is visited, it is checked in the white list. If the URL is found in the white list, but the IP address is different, then this URL is classified as phishing. If the URL is similar but not same, then a warning is given. This tool is also capable of detecting pharming.

Bayesian Anti Phishing toolbar [11] is a Mozilla Firefox extension. It is designed to help users identify phishing websites and protect their sensitive information. It also uses a white list based approach. For a given website, the B-APT toolbar sends the URL to the DOM analyzer. The DOM analyzer then checks if the given URL is present in white list. If it is in the white list, it means it is a legitimate website. If it is not, then DOM analyzer tokenizes the given website and then forwards the tokens to a scoring module. This scoring module consults the database of tokens and computes a score. If the score exceeds a pre-set threshold, then the URL is classified as phishing.

AntiPhish [12] is a Mozilla browser extension that aims to protect inexperienced users against spoofed website-based phishing attacks. AntiPhish keeps track of the sensitive information of a user and generates warning whenever the user types sensitive information into a form on a non trusted website. Most of the browsers such as Mozilla have functionality that allows form content to be stored and automatically inserted if the user desires. AntiPhish takes this common functionality one step further and also tracks where this information is sent. It also stores the mapping of where this

Digital Object Identifier: 10.4108/ICST.COLLABORATECOM2009.8310
http://dx.doi.org/10.4108/ICST.COLLABORATECOM2009.8310

information belongs to. However effectiveness of this approach is dependent on the user.

SpoofGuard [13], [14] is a browser plug-in that uses domain name, URL, link and image check to determine if a given page is a part of a spoof attack. It applies the following three tests to a given page and then combines the result using a scoring mechanism: it uses a stateless method that determines whether a downloaded page is suspicious, stateful method that evaluates a downloaded page in the light of user's history, and method that evaluates outgoing HTML post data. The final score determines whether the plug-in should alert the user and also determines the severity of the alert. False alarm rate depends on how frequently the user establishes new accounts and how frequently the user clears the browser history cache. Many of the unnecessary warnings are the result of frame or redirection problem. If the user opens a new account, and uses same password as another account, it will produce an unwanted warning. Some of the tests compare passwords that are used at a particular site to passwords that were used at previous sites. An attacker could fool these tests by breaking the password field (on the spoofed page) into two adjacent fields, that would look contiguous, but would cause the test to fail. Some of the tests compare images on phished pages to the image that appear on legitimate pages. An attacker could circumvent these tests by slicing an image into adjacent vertical slides and presenting these slices one next to the other.

CANTINA [15] uses a content-based approach to detect phishing web sites. It combines a term frequency-inverse document frequency (TF-IDF) algorithm with other heuristics to determine whether a given website is a phishing one. It uses fives words with the highest TF-IDF weight on a given website as a signature and then submits those five words to the Google search engine. If the URL of the site is found within top results, then that URL is classified as legitimate, otherwise phishing. An attacker could circumvent CANTINA using several techniques. One technique would be to use image instead of words in a given page. Another technique would be to add invisible text, text that is tiny or matches background color of the page. Yet another technique would be to change a lot of words in order to confuse TF-IDF.

IV. Design

CASTLE is a decentralized network tool to detect the phishing sites based on two key components: URL string and content. The key feature of CASTLE is that it uses multiple phishing verification servers (PVCs) to determine if the queried URL is phishing. All PVCs are assumed to be trusted and are connected via a social peer-to-peer (P2P) network. The P2P network is socially administered to prevent attackers from inserting their servers as PVCs. That is, PVCs can be inserted into the network only if a certain number of administrators of existing servers approve the administrators of the enrolled server. This out-of-band trust requirement can be enforced at different strengths based on the threat posed by intruders [16].

A. Architecture of PVC

A PVC is responsible for a certain domain or domain prefixes. For example a certain PVC could be responsible for domain name `cnn.com` or `cn*.com` i.e., it will handle all URLs with prefix `cn` and with TLD `com`. It is possible that a PVC experience heavy workload because it is handling a popular domain. Therefore to balance workload, a PVC may consist of clusters of servers, which would be managed by load balancer. A PVC has two tables: decision table and routing table. Construction of the decision table is based on the content analysis module and administrators of the PVCs. Content analysis module will identify whether the queried URL has similar content to the one hosted at trusted URLs. If the content is similar, then the content analysis module will put that queried URL inside the gray list of that PVC, and administrator of this PVC will be notified about this new URL in gray list via email. Administrators of a PVC will investigate the gray listed URLs and make decision about moving the URLs to other lists such as black list and white list. If the administrator of the PVC does not take any action within a specified period of time, then the entries is automatically moved to black list.

Decision table contains the following lists:

- Black list contains all "active" phishing URLs
- Archived black list contains all phishing URLs which are more than 5 days old
- White list contains all trusted URLs
- Gray list contains all URLs suspected for phishing
- Unknown list contains all URLs whose contents are similar to the content of the URL handled by one of the PVCs in the network and the corresponding URL handled by that PVC.

As mentioned earlier, the average up time for a phishing websites is around 4.5 days [4]. So, to prevent the blacklists getting longer because of inactive phishing URLs, we have *time to live* (TTL) values for each entry in the blacklist. The value of TTL will increase by one every day. Each time, a blacklisted URL is visited, the TTL for this URL would be resetted to zero. If the value of TTL reaches 5, that this URL is moved to the archived blacklist. If a URL present in the archived blacklist is visited, then the URL is moved back to the black list. While making decision about a URL in a PVC, archived black list would not be used in making that decision even if it contain the queried URL; however we would only check archived black list in the background in order to see if any URL (inside archived black list) need to be moved back to black list.

A routing table includes a list of IP addresses pointing to other PVCs. A subset of the PVCs pointed to by the routing table would have a ability to update the routing table. These PCVs are PVCs that are highly trusted by the local administrator (i.e., socially connected).

B. URL Routing Mechanism

The routing table is used to push queries through the P2P network of PVCs. We use only the domain name and TLD

Digital Object Identifier: 10.4108/ICST.COLLABORATECOM2009.8310
http://dx.doi.org/10.4108/ICST.COLLABORATECOM2009.8310

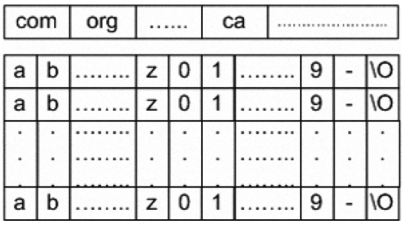

Fig. 1. The structure of routing table

part of URL for routing. Therefore each routing table consists of two data structures: a one-dimensional array (TLDArray), which stores the list of all possible top level domains (TLD), and a two-dimensional array (DNArray), which stores all possible characters of the domain name of the URL as well as the null character "\0". A domain name would be stored with the null character at the end. For example `cnet` would be stored as `cnet\0`. If the URL contains an IP address instead of a domain name, the IP address would be stored in hex format. The structure of routing table is shown in Figure 1. The TLDArray contains pointer to other PVCs which handle domains with different TLDs than the TLD of the current PVC. The DNArray contains pointers to other PVCs which handle domains with the same TLDs as the TLD of domain handled by the current PVC. First row of the DNArray represents the first letter of the domain name handled by the PVC along with pointers to other PVCs handling domain name whose first letter is not the same as the first letter of the domain name handled by the current PVC. The second row represents the second letter of the domain name handled by the current PVC along with pointers to other PVCs handling domain name whose first letter is the same as the first letter of the domain name handled by the current PVC and second letter is not the same as the second letter of the domain name handled by the current PVC and so on.

Figure 2 shows an example of the routing table of PVC handling `cnet.com`. The entries in the TLDArray represent all possible TLDs, whereas entries in DNArray (in bold) represent the character of domain handled by this PVC. It shows the list of entries to which this PVC (handling `cnet.com`) point to. As can be seen in the same figure, TLDArray contains pointer to PVCs which handle domains with different TLDs than the TLD of domain handled by current PVC. DNArray contains pointer to PVCs which handle domains with the same TLD as the TLD of domain handled by current PVC. The first row of DNArray contains pointer to PVCs that handled those domains which do not share any prefixes with the domain handled by the current PVC. Second row contains pointer to those PVCs that handle domains whose names start with `c` and third row contains pointer to those PVCs that handle domains whose name start with `cn`.

We should also be able to add new entries (IP addresses of other PVCs) to the routing table of the current PVC. Entries containing the IP addresses of PVCs handling domains with different TLDs than the TLD of the domain name

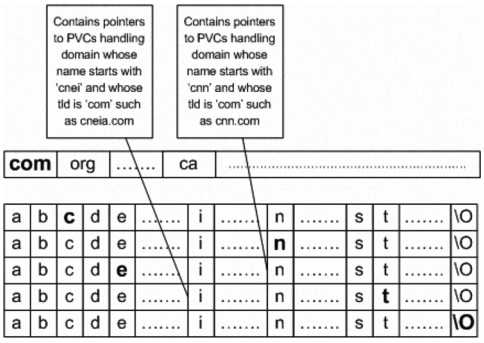

Fig. 2. Routing entries for PVC handling cnet.com

handled by the current PVC will go to the TLDArray, and entries containing the IP addresses of PVCs that handled those domains which does share any prefix with the domain name handled by current PVC will go to the appropriate rows of the DNArray. Assume we want to add the entry containing IP address of PVC handling `cneefq.com` to the routing table of PVC handling `cnet.com`. After adding the entry for `cneefq.com`, the letter `e` in the fourth row of the DNArray in routing table for PVC handling `cnet.com` would contain pointer to the PVC handling `cneefq.com`. Pseudocode for insertion algorithm is shown in Algorithm 1. Whenever we want to insert an entry for a URL in a PVC, we will call insert(URL, IP address of the PVC handling this URL).

Algorithm 1 insert(URL, IP Address)

```
host ← domain name of URL
domain_tld ← tld of domain handled by this PVC
length ← length of host
tld ← tld of URL
domain ← domain of handled by this PVC
determine i such that TLDArray[i] = tld
if domain_tld ≠ tld AND url is not in ip address format then
    add IP address to TLDArray[i]
else
    for i = 1 to length do
        find j where array[i][j] = host[i]
        if host[i] ≠ domain[i] then
            add IP address to DNArray[i][j]
            break
        end if
    end for
end if
```

This URL routing mechanism would help us to determine whether the queried URL is phishing. If there is a PVC which handles this queried URL, this mechanism would ensure we route to the PVC handling that queried URL. If none of the PVC handles the queried URL, then this mechanism would route to the PVC who handles domain which shares the longest prefixes with the queried URL. Afterwards, we will check if the queried URL is present in four lists in the decision table in following order: white list, gray list, black list and unknown list. If it is present in the black list, white list or gray list, the user would be informed about the decision based on the lists in which it is present. If it is present in the unknown list, corresponding URL would be extracted and we would then route to the PVC handling that corresponding URL in order

Digital Object Identifier: 10.4108/ICST.COLLABORATECOM2009.8310
http://dx.doi.org/10.4108/ICST.COLLABORATECOM2009.8310

to find out about the decision. If it is not present in unknown list, content analysis would be performed, which would return a URL (CA URL) whose content is similar to the content of the queried URL. We would store this queried URL and CA URL in the unknown list. Afterwards, we would route to the PVC handling that CA URL. Then we would search for that queried URL inside the three lists of that PVC: white list, gray list and black list. If the queried URL is not present in these three lists, then the user would be informed that decision could not be made. However in the background, search would be carried out in the archived black list. If the queried URL is found in the archived black list, then this URL would be moved back to black list; so that next time if someone tries to query this URL again, this URL would be found in the black list and URL would be returned as phishing. Then owner of that PVC handling URL with the similar content as the content of the queried URL, would have an option of putting that URL in the appropriate list. In this situation, there would be two PVCs in the system handling such a URL: one for content similarity and the other for URL similarity. Pseudo code for search algorithm is shown in Algorithm 2. Whenever search is initiated in order to determine whether the queried URL is phishing, function search(0,null,URL,IP address to which one is connected) is called. It will return 1 if URL is not phishing, 0 if URL is suspected and -1 if URL is phishing.

Algorithm 2 search(index, final_URL, URL,IP address)

```
tld_domain ← tld of domain handled by this PVC
host ← domain part of URL
domain ← domain handled by this PVC
tld ← tld part of URL
determine i such that TLDArray[i] = tld
if tld_domain handled by this PVC ≠ tld AND
url is not in ip address format then
    IP address ← get ip address from TLDArray[i] randomly
    search(0, null, URL, IP address)
else
    for i = 1 to length do
        find j where array[i][j] = host[i]
        if host[i] ≠ domain[i] then
            IP address ← get ip address from DNArray[i][j] randomly
            if IP address = null then
                if queried URL is present in white list then
                    return 1
                else if queried URL is present in gray list then
                    return 0
                else if queried URL is present in black list then
                    return -1
                else if final_URL ≠ null then
                    put final_URL in gray list
                    return 0
                else if URL is present in unknown list then
                    CA_URL ← extract corresponding URL
                     search(0,URL,CA_URL,IP address of current PVC)
                else
                    perform content analysis model and get CA_URL
                    store URL along with CA_URL in unknown list
                    search(0,URL,CA_URL,IP address of current PVC)
                end if
            else
                search(i, final_URL, URL,IP address)
            end if
        end if
    end for
end if
```

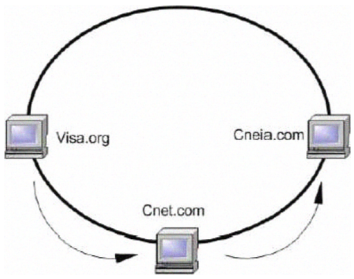

Fig. 3. Hypothetical example showing how routing would take place from PVC handling visa.org to PVC handling cneia.com. In this case, we assume search starts from visa.org

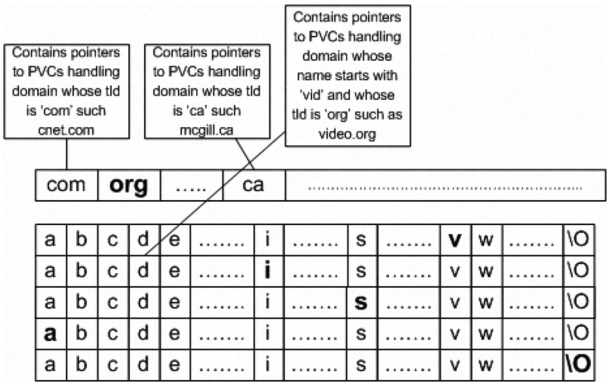

Fig. 4. Routing entries for PVC handling visa.org

C. Example of Routings

Example 1: The Figure 3 shows how we could route to the PVC handling `cneia.com` from the PVC handling `visa.org` assuming routing entries for PVC handling `visa.org` are as shown in Figure 4. In this scenario, we will first check if TLD of the queried URL is the same as the TLD of the domain handled by the PVC handling `visa.org` . Since it is not the same, the IP address of the domain with the TLD `com` would be extracted from TLDArray. It could be possible we extract the IP address of the PVC handling `cnet.com`. So in this case, we would route to the PVC handling `cnet.com`. Then we determine the longest prefix shared by `cnet.com` and `cneia.com`. In this case it is `cne`. Since the fourth character is not the same, we extract one of the IP addresses stored in the fourth row and column containing character "i" of the DNArray. Extracted IP addresses would be the IP address of PVC, which handles domain whose prefix is `cnei`. In this case we route to the PVC handling domain `cneia.com`.

Example 2: Suppose we are querying the URL `cneibbc.com` whose content is similar to the content in `rbc.ca` and currently no PVC has an entry for `cneibbc.com`. Assume `cneia.com` routing table entries are as shown in Figure 5. Figure 6 shows how routing would take place in such a case.

Example 3: Suppose we would like to search for `cneiaf.com`, and assume there is no PVC which handles `cneiaf.`

Digital Object Identifier: 10.4108/ICST.COLLABORATECOM2009.8310
http://dx.doi.org/10.4108/ICST.COLLABORATECOM2009.8310

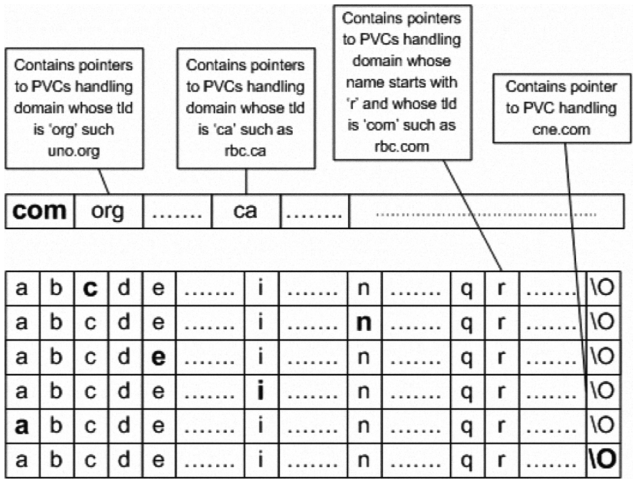

Fig. 5. Routing entries for PVC handling cneia.org

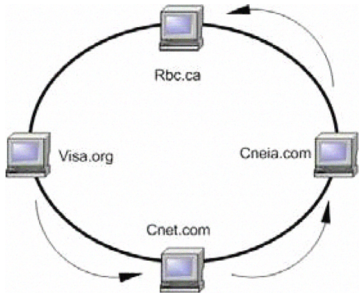

Fig. 6. Hypothetical example showing how routing would take place if we want to determine whether cneia.com is phishing. In this we assume content in cneia.com is similar to the content in rbc.ca

`com`, but there is a PVC handling `cneia.com`, we start the search from the PVC handling `visa.org`. Considering the routing procedure described in *Example 1*, we end up in the PVC handling `cneia.com`, since there is no routing entry for PVC handling `cneiaf.com`. Consequently, we check its own decision tables to make an appropriate decision about `cneiaf.com`. If we are not able to reach any decision, we invoke the content analysis module.

D. Content Analysis

Content analysis is another component of our system. It helps to find out which URL is pretending to look like one of the trusted URLs. A URL is assumed to be trusted if there is a PVC which handles this URL. This can help us to determine the target of the phishing URL and then we would inform the PVC handling that target URL about possible phishing attack. The PVC, which has been informed about the target URL, would have an option of determining whether it is phishing or not.

To do the content analysis, our system has a CA (content analysis) module. CA takes both text and image snapshot of the given URL by rendering them through a browser using one of the webkit based tools called CutyCapt [17]. Then inside the CA we have two sub CA module called TCA (textual CA) and VCA (visual CA). The TCA will use the text snapshot of the URL and tries to randomly extract five sentences from the text snap shot. We call those extracted sentences the textual signature of the URL. Then we push these sentences one by one into the search engine and find out the top five URLs (returned by the search engine) for each sentence. A table is used to keep track of these 5 URLs. We will call the top five results returned by the search engine for each sentence the "search set". Based on the result returned by the search engine, score would be computed using a scoring mechanism mentioned below. However before computing the textual signature, we will determine if the given URL contains the copyright sentence. The copyright sentence must contain one of the following sets of keywords:

- "CopyRight" and "All Rights Reserved" in the sentence.
- "©" and "All Rights Reserved" in the sentence.
- "©" and "CopyRight" in the sentence.
- "©" or "CopyRight" with any numbers in the sentence.
- "Trademark" and "Registered", or "©" and "Trademark" in the sentence.

If the given URL has more than one sentences with the above characteristics, we will return that URL as a suspected one and we will put it into the gray list. If it has only one sentence with the above characteristics, we extract this sentence and remove the date (inside the sentence) and feed such a sentence into a search engine. We will call the first URL returned by the search engine the "copyright URL". The reason that we remove the number from the copyright sentences is that some phishing websites change the date of the copyright sentence order to index differently from the legitimate websites in search engine. Then we will compute the textual signature. We feed each of the five randomly selected sentences separately into the search engine. Score of one would be assigned to the URL returned by search engine for each sentence if it is present in the top five results. The score would be increased if the search engine returns the same URL for the other sentences. Then we will check the presence of the "copyright URL" in every search sets. If the "copyright URL" is not present, we will repeat the above procedure of computing the textual signature up to five times. If the "copyright URL" is still not present in the search sets, we return the queried URL as a suspected URL and we would put it inside the gray list of the appropriate PVC. If the "copyright URL" is present, the "copyright" URL would be assumed as the target URL.

If the queried URL does not contain any copyright sentence, then the URL with the highest score would be returned as the target URL. After determining the target URL, we will route to the PVC, which is responsible for target URL and we will put the queried URL into the gray list. Figure 7 shows the flowchart for the content analysis module.

Digital Object Identifier: 10.4108/ICST.COLLABORATECOM2009.8310
http://dx.doi.org/10.4108/ICST.COLLABORATECOM2009.8310

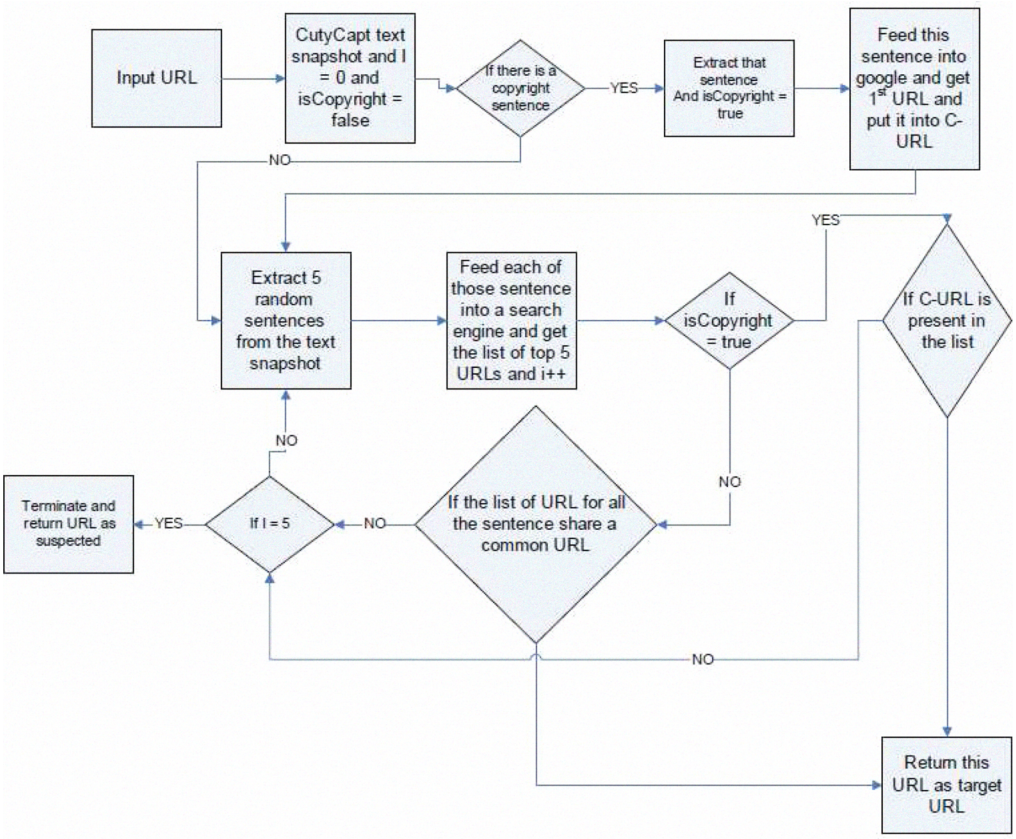

Fig. 7. Flowchart for content module

V. Complexity Analysis

For simplicity purposes, we will refer to domain or domain prefixes for which PVC is responsible as *h*-domain (hosted domain) or *h*-domain prefixes respectively. We know in each routing step, either a PVC forwards the search request to other PVCs whose *h*-domain share a prefix with the domain name of the queried URL or a PVC forwards the search request to a PVC handling domain with different TLD.

In order to go to a PVC handling a domain with different TLD, it takes only 1 hop since we pick up the IP address of this PVC from the TLDArray of the current PVC and then route to this PVC. In order to go to the PVC handling a domain name whose first letter is the same as the first letter of the h-domain of the current PVC, it takes at most 2 hops since we extract the IP address of those domains who share the first letter with the h-domain of the current PVC from the first row of the DNArray of the current PVC and then we route to it. In order to go to PVC handling domain name whose first two letters are the same as the first two letters of the queried domain, it takes at most 3 hops.

So in order to go to the PVC handling domain whose first n letters are same as the first n letters of the queried domain, it takes at most $n+1$ hops. As mentioned in the Section IV, the NULL character is also consider part of domain name, so we will make use of the NULL character to restrict the number of routing hops to the domain length. For example if domain name is `cnet`, in our design it would represented as `cnet` plus NULL character. In this case it would take at most up to 6 hops assuming cnet.com is handled by one of the PVC since there are 5 letters including NULL character in the domain name. If the queried URL is stored in any of the PVC's decision list and assuming URL length is n,then in this case it can take up to $n+2$ hops to reach that PVC.

As mentioned in Section IV, it is also possible that the URL is not handled by any PVC and is not present in white list, gray list or black list of all the PVC. In this case, we would route to PVC whose *h*-domain is similar to the queried domain. After routing to this PVC, we would use unknown list or content analysis module to find the domain whose content is the most similar to the content of the queried domain, and then we would route to the PVC handling domain which is stored in the unknown list or the URL returned by content analysis module. For this scenario, assuming m is the length of the domain and this domain is handled by one of the PVCs whose content is the most similar to the content of queried domain, it would take up to $m+n+4$ hops.

VI. Security Analysis

Security threats to CASTLE can be categorized into two types: threat to the content analysis module (CAM) and threat to the social network of PVCs.

Digital Object Identifier: 10.4108/ICST.COLLABORATECOM2009.8310
http://dx.doi.org/10.4108/ICST.COLLABORATECOM2009.8310

A. Attacks against the CAM

Attackers can put invisible text in order to circumvent the content analysis module of CASTLE because sentences which are part of the invisible text are used by CAM to analyze the content. As a result, CAM can return a URL whose content is not similar to the visible content of the queried URL. So our CAM module would put this queried URL in the gray list of the PVC handling domain whose content is not similar to the visible content of queried URL. However administrator of this PVC would notice it; so he/she would put it inside black list. Thus attackers cannot circumvent CAM by inserting invisible text.

Also since we are dealing with the exact domain name of the visiting URL, all the misleading information used in subdomain names to mislead the users would be useless. For example if the URL is *www.ebay.online.com*, the user may think that this URL belongs to *eBay*, but in our design, it would be treated as *www.online.com* not as a subdomain of *eBay*.

In addition, all the changes in the user's browser such as hiding the address bar, faking the address bar, or displaying the URL in unicode format would be useless because we are dealing with the actual domain who hosts the websites and the content of the phishing website in order to find the website that the attacker is trying to phish.

B. Attacks against the social network of PVCs

Another type of attack is against the P2P network of PVCs which handle the information regarding the decision that should be rendered for a given URL. In this case we assume that all the PVCs are connected through the social network and propagation of the routing information will be taking place over the social network. So the only attack against this would be a intrusion into the social network. An attacker can compromise a node in the social network even without the awareness of the authorities of that PVC and may try to give false information to other PVCs who may ask the compromised PVC. But in this case an attackers can only propagate wrong routing information for the domain they are responsible. We call that node a "bad node" inside the social network. In general bad nodes can do the following actions inside the social network: ignore queries, misroute queries, propagating false routing updates, refuse to store or return items and return incorrect items. To address these issues, we assume that at least one node in the social network is alive and not malicious. Then we try to prove that a node is bad with some mechanism that will result in the eviction of such a node from the social network.

We introduce an online certificate authority (*OCA*) that issues a certificate to each node upon it joining the network. Also each node must inform the *OCA* about the address of two possible backup nodes in case of failure or attack. Back up nodes would be independently maintained and updates of the routing information will be done independently. Each node must know the private key if it wants to propagate routing information to other nodes. Also each node must renew its own key after its expiration of the key. So if a node proved itself to be malicious, the *OCA* would refuse to issue a certificate to it. Thus it would be evicted from the social network. So any attempt from anyone inside the network to say something inconsistent will result as misbehavior and eviction from the social network.

Misbehavior can be established through other mechanisms as well. A node can periodically check its information with its connected peers. Connected peers are neighboring nodes according to the the routing table. So each node will ask connected nodes about the information that it has in its black or whitelist and monitor the responses of the connected nodes. Connected nodes can be responsive or not responsive. If they are not responsive, then there should be some problem with them or they might have been hijacked by attackers who do not want to respond and just want to give out false information about the current PVC's domain. In this case, the node who has checked the unresponsiveness will inform the *OCA* about the connected nodes which are not responsive. If the number of complaints for a certain nodes exceeds a threshold, then that node would be tagged as a suspected node and the *OCA* will perform its own procedure to validate the complain. The *OCA* will select the random number of PVCs in the network excluding the ones which have complained to ask the suspected node about its information. The reason that we choose random nodes inside the network is because we do not want to show the suspected node that it is on probe. If the queries come from different sources, this can be considered as a normal query process inside the network and can not be detected as a probe process. After the random nodes checked the suspected node about its information to see whether the information that it supplies is correct or not, they will inform the *OCA* about the correct or incorrect answers returned by the suspected node. If the number of incorrect answers exceeds a threshold, then the *OCA* will tag that node malicious and will inform its immediate back up nodes, so that those backup nodes can revoke their links with that malicious node and take the responsibility of that node inside the network. Also the *OCA* will not issue to that node a certificate anymore; so no more queries will be forwarded to that node anymore and it will be evicted from the network at the time of renewal of the certificate for good. The reason that we ask the *OCA* to evaluate the complains by itself is because we want to avoid the case where some nodes join together in order to evict one of their competitors from the network by providing false information to the *OCA*.

To protect the *OCA*, we can also consider a total distributed-limited state shared replicas for it. Also only nodes that are currently part of the network need to inform or ask the *OCA* so we can filter all other traffic to it.

VII. Implementation

We have implemented CASTLE in JAVA. CASTLE comprises of 2000+ lines of code as well as some modules such as CutyCapt, Google AJAX Search API, JAVA RMI, and JSON. We used CutyCapt to get the text snapshot of the content of

Digital Object Identifier: 10.4108/ICST.COLLABORATECOM2009.8310
http://dx.doi.org/10.4108/ICST.COLLABORATECOM2009.8310

a web page. Google AJAX Search API was used to access the Google database to search for a particular sentence and to retrieve the corresponding URL. JAVA RMI library was used to build P2P networks, which connect PVCs. Content analysis module has a simple command line interface, which returns a URL either labeled as phishing, suspected as phishing, or legitimate. Each PVC also has a command line interface where the administrator of PVC can modify the routing table or the decision table. It should be noted that the prototype is still evolving and new features will be added in the future.

VIII. Experimental results

We tested content analysis module and routing algorithm separately

A. Routing algorithm

We tested our routing algorithm on planet lab with 100 machines. In our experiment, we had 100 PVCs (each machine representing one PVC) and each PVC handled one URL. About 35 URLs had TLD com and about 20 of URLs had TLD ca. Black list, white list and gray list in each PVC was populated with 7 URLs. In total, each PVC was populated with 21 URLs. Since we encountered scalability issues on Planet-lab, we could not test on more than 100 machines.

In the first part of this experiment, we carried out a search for all 100 URLs from four different PVCs: each PVC handled domain with different tld. We repeated this experiment 15 times. The minimum response times are shown in Figure 8. As shown in these figure, most of the time, times for a search are about 2 seconds. However in few cases, times are 18 seconds, which is due to the abnormal behavior of some of the nodes inside the PlanetLab.

In the second part of this experiment, we carried out a search of 2100 URLs stored in decision lists of PVCs and 100 URLs from four different PVCs: each PVC handled domain with different tld. We repeated this experiment 7 times. The minimum cumulative response times are shown in Figure 10. As shown in this figure, the response times for the 90% of the search are at most half a second. However in few cases, the response times for 2% of the search are more than 2 seconds, which is due to the abnormal behavior of some of the nodes inside the PlanetLab.

In the third part of experiment. we collected about 350 URLs (which were not handled by any of the PVC and were not present in decision list of any PVC). Then we carried out a search of these 350 URLs from four different PVCs: each PVC handled domain with different tld. We repeated this experiment 5 times. The minimum response times are shown in Figure 9. As shown in these figure, most of the time, the response times for a search are about 2 seconds. However in few cases, the response times are 18 seconds, which is due to the abnormal behavior of some of the nodes inside the PlanetLab.

B. Content Analysis Module

We conducted an experiment which consists of 150 phishing URLs along with 50 legitimate URLs in order to measure the

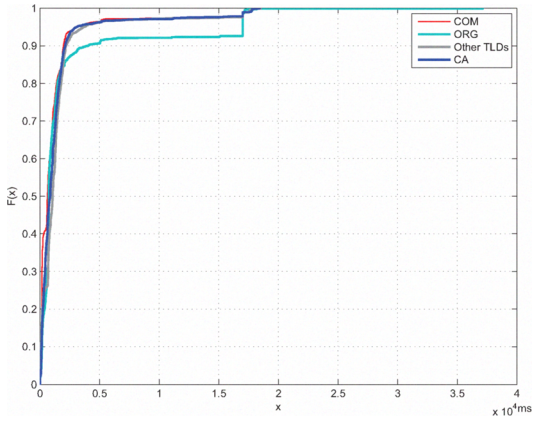

Fig. 10. The response times for 2200 searches carried out from four PVCs: each PVC handled domain with different tld

effectiveness of our approach. From April 5 to April 22 '2009, we collected most of the phishing URLs from `phishtank.com` [18]. These phishing URLs from `phishtank.com` were selected within one hour of being reported. Some of the phishing URLs were also gathered from the user spam email account. We also manually chose 50 legitimate URLs in different categories such as banks, popular e-commerce sites, and top 20 popular sites from the Alexa Web search such as `rbc.com`, `citibank.com`, `ebay.com`, and `cnn.com`. We used a test scenario to gather our results. In our test scenario, we fed all 200 URLs into our content analysis module. In the case of a phishing URL, which is fed into our module, we compared the URL returned by our content analysis module and the actual URL that is being phished by the phishing URL. In a case of a legitimate URL, which is fed into our module, we compared the legitimate URL itself and the URL returned by our content analysis module. In this case, both URLs should be the same. In this case, true positive was 97.5% and false positive rate was 2.5%. To decrease the false positive rate further, we applied a set of heuristics described in section under copyright property characteristic and ran another experiment. After the new experiment was run, false positive rate fell down to 1.5% and true positive rate was 98.5%.

IX. Conclusion

This paper describes a new framework called CASTLE that provides a collaborative approach for building anti-phishing databases. Such anti-phishing databases are at the core of browser-based toolbars, plugins, and extensions that prevent users from falling prey to phishing attacks. CASTLE is scalable to any number of nodes because response time does not depend on total number of nodes. Instead it depend on the length of the domain name of queried URL.

The advantage of CASTLE approach is that it can provide a substrate for integrating many different anti-phishing strategies. For example, the PVCs that are part of the P2P network can employ diverse techniques for making the local decisions. Some PVCs can be using a manual examination based approach while others could be using a combination of

Digital Object Identifier: 10.4108/ICST.COLLABORATECOM2009.8310
http://dx.doi.org/10.4108/ICST.COLLABORATECOM2009.8310

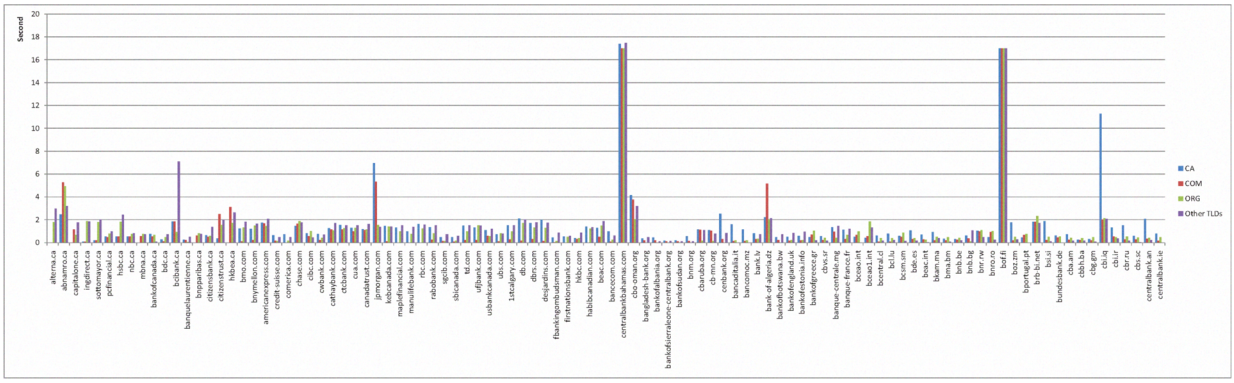

Fig. 8. The response times for search carried out from four different PVCs: each PVC handled domain with different tld

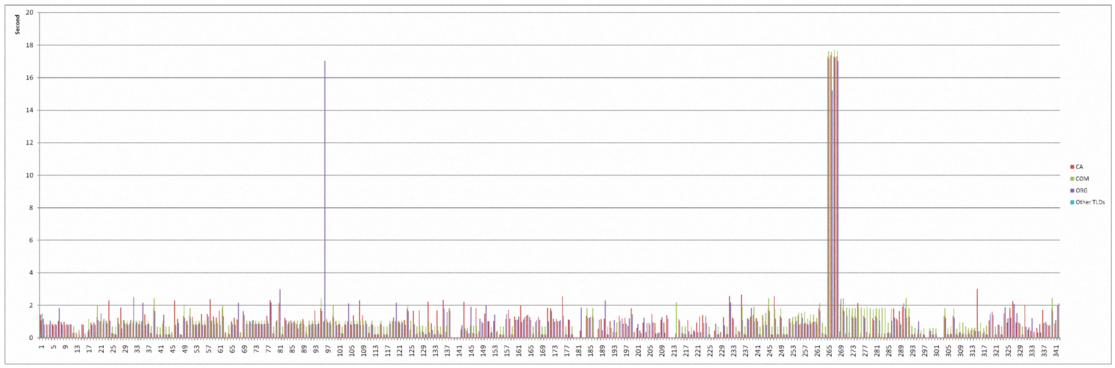

Fig. 9. The response times for 300 searches of unsupported URLs carried out from four different PVCs: each PVC handled domain with different tld

automated analysis and classification and manual examination. The use of diverse techniques increases the scalability of the anti-phishing effort. Some of the anti-phishing approaches mentioned in Section III require human intervention. CASTLE uses a semi automatic approach.

Further, to the best of our knowledge this is the first anti-phishing framework that combines location-based and content-based techniques into a single system. By using the dual approaches, we think zero day vulnerabilities that are faced by other techniques can be handled by CASTLE.

REFERENCES

[1] M. Jakobsson and S. Myers, *Phishing and Countermeasures: Understanding the Increasing Problem of Electronic Identity Theft*. Wiley-Interscience, 2006.

[2] A. Herzberg and A. Jbara, "Security and identification indicators for browsers against spoofing and phishing attacks," *ACM Trans. Internet Technol.*, vol. 8, no. 4, pp. 1–36, 2008.

[3] "Phishing attack trends report - second half 2008," http://www.apwg.org/reports/apwg_report_H2_2008.pdf, Mar. 2009.

[4] A. P. W. Group, http://www.antiphishing.org, Anti Phishing Working Group.

[5] J. Ma, L. K. Saul, S. Savage, and G. M. Voelker, "Beyond blacklists: learning to detect malicious web sites from suspicious urls," in *KDD '09: Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*. New York, NY, USA: ACM, 2009, pp. 1245–1254.

[6] A. Y. Fu, "Detecting phishing web pages with visual similarity assessment based on earth mover's distance (emd)," *IEEE Trans. Dependable Secur. Comput.*, vol. 3, no. 4, pp. 301–311, 2006.

[7] eBay, http://pages.ebay.ca, eBay.

[8] News.com, "Netscape readies antiphishing browser," http://news.cnet.com, 2006.

[9] Microsoft, "Technology overview:microsoft windows internet explorer 7," Microsoft White Paper, 2006.

[10] J. Kang and D. Lee, "Advanced white list approach for preventing access to phishing sites," in *ICCIT '07: Proceedings of the 2007 International Conference on Convergence Information Technology*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 491–496.

[11] P. Likarish, E. Jung, D. Dunbar, T. Hansen, and J. Hourcade, "B-apt: Bayesian anti-phishing toolbar," May 2008, pp. 1745–1749.

[12] E. Kirda, , E. Kirda, and C. Kruegel, "Protecting users against phishing attacks with antiphish," in *In COMPSAC 05: Proceedings of the 29th Annual International Computer Software and Applications Conference (COMPSAC05) Volume 1*. IEEE Computer Society, 2005, pp. 517–524.

[13] N. Chou, R. Ledesma, Y. Teraguchi, D. Boneh, and J. C.Mitchell, "Client-side defense against web-based identity theft," in *11th Annual Network and Distributed System Security Symp. (NDSS'04)*, 2005.

[14] "Spoofguard," http://crypto.stanford.edu/SpoofGuard/.

[15] Y. Zhang, J. I. Hong, and L. F. Cranor, "Cantina: a content-based approach to detecting phishing web sites," in *WWW '07: Proceedings of the 16th international conference on World Wide Web*. New York, NY, USA: ACM, 2007, pp. 639–648.

[16] H. Yu, M. Kaminsky, P. B. Gibbons, and A. Flaxman, "Sybilguard: defending against sybil attacks via social networks," *SIGCOMM Comput. Commun. Rev.*, vol. 36, no. 4, pp. 267–278, 2006.

[17] "Cutycapt," http://cutycapt.sourceforge.net/.

[18] "Phishtank," http://www.phishtank.com/.

Digital Object Identifier: 10.4108/ICST.COLLABORATECOM2009.8310
http://dx.doi.org/10.4108/ICST.COLLABORATECOM2009.8310