Internet & Web Algorithms
Final Project

- Aditya Sinha
  M03186912
- Harshvardhan Kelkar
  M03187173

Phishing: Nuts, Bolts and CounterMeasures

What determines the credibility of a fraudulent website? This is one of the many questions that this expository essay will address. The key for a successful phishing attack is the user-interface. Phishers are responsible for the creation of web-pages with a impressive presence such that it causes the victim to overlook the security measures installed in web-browsers .

There is no way for users to reliably determine if a connection is secure. The strategies used by attackers to con users are based on lack of knowledge, visual deception and lack of attention.Many users do not have the skills to distinguish forged email headers from a legitimate Header. Phishing sites thus exploit the lack of knowledge of users pertaining to operating systems, applications ,emails and how the web works in general. Ignorance About the functioning of SSL certificates is also exploited by Phishers to mount attacks. Furthermore text ,images and windows are mimicked by Phishers to give the website a legitimate look and feel. Images of a legitimate hyperlink are used by Phishers. They can impersonate a rich and fully functional website complete with links,logos, Animated pictures, graphics and fake SSL indicators. The type and quantity of requested information at a website may be the only clue for a end user to differentiate between a legitimate and a rogue website. Users are also more likely to respond to a phishing email that arrived from a friend's spoofed address.

Even if the user is aware of the above mentioned strategies, there is one more way for Phishers to compromise the users information. They do this by exploiting the users Lack of attention to security indicators or to the lack of attention of the absence of security indicators. A spoofed image of a security indicator may be inserted where none exists. Based on a study that was undertaken ,it was determined that users employ various strategies to determine website legitimacy.

These strategies are base upon 1.security indicators in website content, 2.Content and domain name only, 3.Content and address plus HTTPS, all of the above ,all of the above plus certificates. However the methods outlined above are not adequate to give complete protection from phishing attacks. This is because they are vulnerable to exploitation by phishers due to the extent of human interaction involved in the process.

Participants in the study believed that correctness of the URL is the primary criterion by which they decide if a website is legitimate or not. Knowledge about the security padlock icon and HTTPS is limited and unclear in users which makes them vulnerable to Phishing attacks. For instance users may not be aware that the padlock icon is located in the interface constructed by the browser around a web page also known as the "Chrome" Only under specific conditions while icons embedded within the webpage can be arbitrarily placed to induce trust .

A lack of knowledge of how certificates work is also a impediment in protecting users from phishing attacks. The inability of users to distinguish between a Certificate authority signed certificate and a self-signed certificate opens up major vulnerabilities which need to be plugged.

The indicators of trust as presented in web browsers are trivial to spoof. The study illustrates the need to incorporate human ways of thinking in the implementation of a security system and not just traditional cryptography based techniques. There is a urgent need for security indicators to be visible to users under un-secure and un-trusted conditions. The future work proposal is to incorporate identity proof of a remote server in such a manner that it can be verified easily by a human user.

We will now examine a method to detect Phishing websites or the emails which direct users to these sites. This method was evaluated using a set of 860 phishing and 6950 non-phishing emails with a success rate of 96% in the identification of Phishing emails and a misclassification of the order of 0.1%. The method involves the use of a tool known as PILFER.

Phishing attacks are often packaged in emails which seem to originate from trusted entities such as eBay or PayPal. These emails then go on to ask for sensitive information like credit card numbers and passwords. As we can see from the discussions earlier ,many users think of it as legitimate and become victims of Phishing attacks. It is hard for computers to detect phishing attacks as a consequence of the simplicity with with copies of websites can be made. Users are vulnerable to this growing threat due to the ineffectiveness of current filters in detecting these attacks.

The method works by extracting information from external sources as well as from within the email. A feature vector is then defined by the combination of the internal and external sources . A decision based on the feature vector and a trained model is made as to detect a phishing attack.

Browser toolbars were the first type of Phishing filters that have been designed. Even though Browser toolbars have achieved accuracy of 85% ,they have less contextual information when compared with email filters. Even with the toolbar active, user still is part of the decision making process and hence is not completely protected from an attack. eg. If the user in question disregards the dialog box warning , and proceeds to a fraudulent website.

By intercepting and filtering emails which are phishing emails the user can be taken out of the decision making loop ,thus enhancing protection and improving productivity at the same time.

Spam filters till now have been ineffective in filtering out Phishing emails.At the same time there aren't many methods at present to detect phishing emails .The structural features of the email have been incorporated in prior methods along with the presence of 18 keywords and richness of vocabulary ,subject line. The email client Thunderbird looks out for three features namely ,the presence of IP-based URLS Non-matching URLS and the presence of an HTML "form" element. The proposed solution extracts a plurality of features so as to gain more information about a contextual attack and can work in tandem with an existing spam filter.

The approach is one of classification based on machine learning. There are essentially two classes 1. The class of phishing emails ,2.The class of legitimate emails. High accuracy has been achieved by the use of just 10 features.
1.IP-based URLs :A link which refers to a page with a IP address indicates a potential phishing attack.
2.Age of linked-to-domain names: Phishers register similar sounding names to legitimate websites. A WHOIS query performed to see how old a domain name is.If it falls below a threshold the message is flagged.
3.Nonmatching URLs: If the HREF of the link is to a different host than the link in the test URL ,the email is flagged.
4.Here links to non-modal domain
5.HTML emails: A HTML email is flagged down if it contains a MIME type section

6.Number of links: This is a continuous feature and pertains to the number of links in the HTML part of a email.
7.Number of Domains: The number of distinct domains are counted
8.Number of Dots: The maximum number of dots in a URL which does indicate the possibility of existence of a redirection URL.
9.Javascript: Any email containing javascript is flagged.
10.Spam –filter output: Using the ability of existing spam filters based on some predefined thresholds and weights.

A webpage in a browser environment can be classified using most of the features given above .Browser history provides classification information based on the presence or absence of current website in the browser history . Legitimacy of a web site can be ascertained based on a large no of site visits. It is important to make the classification whether browser went to current page explicitly or it was redirected.

In the empirical evaluation of the method, a random forest is used as a classifier. Random Further classification is carried out using decision trees. Ham corpora a publicly available dataset from SpamAssassin and Phishingcorpus also publicly available were used to test the said implementation.

The additional challenges that are faced pertain to control the false negative rate by tackling the missing information. Another challenge is that since Phishing sites are shortlived one need s to monitor the age of a data set. It cannot be said with certainity that the data sets are representatives of peoples email inboxes.

The accuracy of PILFER combined with a Spam filter is better than it working stand alone. It is also to be noted that while working stand alone PILFER generates very few false positives and is successful in eliminating a large percentage of Phishing emails.

Phishing emails have certain unique properties even while being a subset of spam emails. The above discussion described how phishing emails may be detected and filtered and how similar methods can be employed in web browsers. The information available to counter phishing attacks has increased even as phishing attacks themselves evolve and gain sophistication.

We now turn to a more complex scheme which assumes that users will be spoofed ,they will enter passwords at insecure locations and still the implementation of the said scheme would detect and protect users from phishing attacks. The scheme consists of a server side component and a client-browser plug in. It is important to minimize false positives

as they erode confidence in the system and false negatives which in fact allow Phishers to target the system.

Over use of pop ups have rendered them as a ineffective mechanism to warm users against phishing attacks.The current system proposes to protect the user even though they pass through the pop up warnings. There is no blacklist in the proposed system ie no user will be prevented from accessing a existing account.

The implementation detects credentials in the form of triads (uid,pwd,dom) ie userid ,password and domain and generates alert for the browser when credentials from a protected list are encountered.
If a user enters the same uid,pwd at domains different from dom then it may or may not be a cause for concern but if multiple users doing so will indicate to the server that an attack is in progress and then dom responds by blocking all the compromised accounts.

It is the goal of Verification systems to make spoofing attacks impossible or infeasible. Domain specific passwords is one of the ways passwords of user accounts are protected by using the hash of the password salted with the pertinent domain name which would render the Phishers attack futile even if they manage to retrieve a password.

Another technique is to maintain a dynamic blacklist and alert users when they come across a phishing site. Behavior change in a proactive manner based on visual warnings has still not been prevalent amongst users .A static set of rules pitted against an active attacker would not be an effective mechanism to prevent phishing attacks. Typing the password used at a previous site at the phishers site is what distinguishes phishing sites from other sites. The current implementation of solution relies on aggregating data at a server thus aiming to detect the attack at a global scale. Though for the solution to be truly effective the client plug in has to be used by a large number of users.

The credentials as mentioned earlier which are actually stored in a protected list are of form [dom,H1,H2] with H1 & H2 being the hashes of uid and pwd respectively. The protected list is restricted to 256 entries which the authors assume is sufficient to store all important password sites for any given user.To counter any tricks that the phisher may employ key strokes are accessed before they reach the browser and it is checked whether the domain name that has been typed belong sto a whitelist and if not then the event is reported to the server. The server at the same time has a extensive and dynamic whitelist so that legitimate websites are never blocked.

There is a need to distinguish between phishing attacks and password re-use. The data across various sites is aggregated by the server which greatly simplifies this task. Whenever a user enters a password at a non whitelisted site that instance is logged in the server and if many users exhibit such behavior for a non whitelisted site ,that site is added to the UniquepwdRqd list. This means that protected list passwords cannot be entered at that site.

Flushing attacks are prevented at client side by checking that a POSTed password was actually typed. The password is assumed to satisfy at least one of 3 mentioned criteria only upon which the solution is implemented.

The implementation considers various scenarios where the user might inadvertently use a password at one account into another. In such a case alert is not generated as the implementation is intelligent enough to know that a single instance is not indicative of an attack. Problems arise when user visits machines other than their own. In such a case the users protected list is unavailable thus leaving user vulnerable to phishing attack. Secondly , user may leave hashes of the user's account details on the machine.

Thus this is the only major vulnerability in the scheme that users at random internet kiosks will not get protection. But due to data aggregation capabilities of the aforesaid scheme a server is able to detect attack and respond to it in a timely manner.

Privacy concerns are addressed by obfuscating the domain names .This is interesting as as similar scheme had been talked about in "Invasive browser sniffing and countermeasures " by Markus Jacobsson.
Though the system is extremely effective in blocking out phishing attacks it does have a few loopholes. If a screen keyboard is used the system would be ineffective against it ,so would a email asking user to call at a particular number,if the user actually goes ahead and calls that number.

A phishing attack may be mounted on the system if the clients are refrained from sending information to the server.Another concern is if the attacker is successful in preventing server from detecting attack by hiding below a suspicion threshold that it may have. Two more Security flaws are if a attacker mounts a DOS attack onto a legitimate site using the system or if the attacker manages to access password by examining hash at client end.

Thus we have examined the said system which neutralizes most phishing attacks by making effective use of hashes which are stored at client side and reported to the server whenever sensitive information is entered at non-whitelisted sites.

We see that there is strong correlation between the paper's that we chose which became evident as we progressed and summarized their results. So now that you know all that you need to know about Phishing , we concentrate on protecting the browser , the client in essence you . Every time you access a site your cache is updated with their "stuff" - pictures , pages , media content , tons of "stuff" they want to keep in "memory" about you . This feature as such is what makes browsing faster than it could have been and gives it a more personalized touch to it. So what do I want ,say if I was a server . I want to harvest selected internet browsing history from various clients (read users) . Ironically one way to preserve the client privacy is to do it via a particular method we know is a major security vulnerability in the client agents .A vulnerability that I shall discuss how to "plug" later on in the paper. For now , we have a vulnerability that we are going to change to a security feature. The bad turns good. Ideally you don't want to run something on the client to do what you need to do . This so because of two reasons . Firstly we cant be sure that everyone uses it . Secondly even if we were sure that everyone uses it , every client agent needs to be changed , which is not really practical . So we go for a solution that can be implemented in the server side .
I , the server , can benefit a lot if I know what kind of sites you have been to , serve you better , know if you have made a fault , know if you are about to make a fault . Let me explain , on all of our browser we have histories stored . Typically these are for nine days , but they can vary depending on the user settings and what kind of browser that you are using . These histories can be "exploited" to let the devil (read phishers) or angels (read us , the good guys) know where you have been . Think of the good ways that you can use this feature to your advantage . A financial institution , if they came to know that you went to one of the sites (they don't have to know the particular site) that could be possibly a phishing site then they could warn you . If it was a commerce site then knowing where all you went might help them serve you better , give you better recommendations . Plethora of benefits can be found if we see it in a good way . Here is how this could work in short . A third party carrying a list of X sites goes to the client , finds out in a boolean function if you have been to any one of the sites in X or all of them . That is all .It need not know what site you went to. If this were the list of all the phishing sites then you might have a problem and thus could be warned. That is how it works broadly . Before we go into the details of it all let us see the ethical qualms that we can have about all of this . For such a privacy preserving history mining process if you think about it , it does not tell you what site you went you . It just tells you what kind of site you went to . On the looks of it , something about the user is made known but then if there was a phishing

attack on your system that you had no clue about , I am quiet sure you would be grateful if someone pointed it out to you . Also it is not as if this service is going to be run without telling the user . The server or servers in question want to implement this particular feature can have an option given to the user that if you want us to serve you better or help you catch phishers and make you more secure let this feature be , if not you are welcome to do things on your own . So we can see this feature to be an optional feature .

Let us see in detail how this works . A client C goes to a sever S . C already has a history H that has a list L that consists of all the sites that you have visited over the last some number of days . >From now on we assume that this user has given the server S the permission to run this particular feature for him . Now the server uses *cgi* scripts to work this feature . The server loads a list of known sites say K . Note here that this list is dependent on the kind of the service that is being provided by the server . If the server is a commercial portal than more often than not it would be a list of related pages that host articles , objects that the user can buy . If this server is a security or a financial portal then it could be a list of phishing sites . The *cgi* script that runs this server does not load what all sites you went to , rather it works on the *AND , OR* principle . It would compare the K against the L . Whatever sites that have been visited would be shown as of different color owing to the feature of CSS . Now the script on the server S does either a AND operation , in which case it would report back whether all of the sites in K have actually been visited or not , or a OR operation , in which case it would report back whether any of the sites in K have been visited or not . Another feature that could be part of the same feature could be a fuzzy script that does neither a AND or a OR , but relies on a variable *k* , that is if *k* number of sites from K and L match then only report back . So now we come to the part where the report has been sent back to the server . Based on the reply from the script the server knows for instance whether you have gone looking for blue tooth headsets to buy or gone looking at phishing sites . Based on this it could remember cooler and better blue tooth headsets or tell you that you have been stupid and that you have been to phishing sites and you should not do it again !

Having said ,the feature that you can know whether you have been to a particular site is not really something that you as a user would like to have . Anyone who can code html , cgi scripts can get to know where all you have been if you have not cleared your cache , history . Imagine if I was a phisher and I knew that you have been to say www.orkut.com I know then that you have an account there . I could design a page – know this that when I say design a page I mean just copy the source code of that page , send it to you with a different name say 0rkut.com . Seems pretty much the same right ? orkut and 0rkut , the o is replaced with a 0 (zero) . It would look exactly the same .Only now when you click the submit button , I would have inserted code there that would send me your user name and your password and then like a good boy I would redirect you to the original site . Voila !

you would go where you thought you should be going , I would from now on have access to your account . Everyone's happy . Except the next time you log in you would see talking utter nonsense about yourself and giving out details about yourself that you didn't intend to . Not so happy then are you . That is why they say never click on links that you are not sure about . I never do , but then not everyone is as smart as I am . So we need to figure on a way where we don't depend on the attentiveness of the user . Protect them from themselves . I have now established grounds for making sure that this feature is "taken care of" . I am now going to discuss two methods that we can take care of this vulnerability .

Before we proceed it is worth mentioning that either of the following methods don't take care of another specific type of attack – control timing attack . Whenever a server wants to display something on a user's agent it sends a control message which probes the cache / cookies to see if the user has been to its site before or not . A person can time these control messages to see if the user has indeed been to the site or not . If the user already has been to the site before and has information in the cache the control messages would take a lot of time as they have to find that pertinent file in the cache . If it is not then the control messages would do their work quickly .

The first method is to distort data , throw in garbage , in short make sure that even when someone reads your cache , history it is perfectly meaningless . Webpages broadly can be divided into two parts – internal and entrance . Entrance webpages are the pages that you type in your address bar to go to . Like typing in mail.yahoo.com would make it an entrance webpage . An internal page is something that you are sent to from the entrance page that you went to or from another internal page that you were already on . Like go to google.com , search from something . After getting the results you could go to another site that would then make that site an internal page . Note here that entrance and internal URL's don't really have a clear difference . For example , mail.yahoo.com can be an entrance URL when you type that in and go to that page or it could be the internal URL if you went to the entrance URL yahoo.com and clicked on the mail link there . We make changes in your cache and history such that both of this class of URL are secured . The feature of knowing sites from the CSS has one problem – it has to be an exact match , www.x.com/inkipinki would match only www.x.com/inkipinki and not www.x.com/inkipinkiponki or even www.x.com/onkiponki . That is the problem of the feature that we are going to concentrate on and make a counter measure .

As I pointed out for the first method we can not be sure that the user is going to do anything for himself , even though it is for his benefit . We need to come up with a way to implement a server side solution that would work for all of the users . This ways we make sure that every body is benefited for sure . Here is the actual method that we are going to do the needful , for the entrance URL we throw in a list of URL's which are

going to be of the same type as the one that the user went to . For the internal page or media content , we append the URL with a random number at the end of the URL . As I mentioned it has to be an exact match or else it would not work , so if we had a random number at the end we would have made sure that it would not match . Now is a good time to go in for the details .

We have again a client C that goes to the server A . Now before it actually goes to A we have something called a *translator* in between .All of the stuff that I spoke about in the preceding paragraph is actually done by the *translator* . For the entrance page , the translator would send back a list of all of the sites that are of similar nature to the site that you are actually going to . So if you went to one banking site , it would seem that you have been to a bunch of banking sites . Any phisher trying to "read" content off your cache/history would be baffled that you went to 50 bank sites .He would not know which page to copy , which bank you have an account in . Though if we assume that the phisher is dumb , he would think he got lucky and found a multi millionaire who has an account in 50 banks ! In any case if he sends you back a link of a bank page the chances that he would be right is about is 1 in 50 or about 2 % .Increase the amount of garbage URL we decrease the probability even further . That was the entrance URL strategy . As for the internal page , the client C sends the http request to the server S . S checks whether it is a html page that the client is looking for , if it is with a random number already generated at the end then the sever checks if it matches to the number stored for that client C . If it is then it responds to that request . If the numbers not present then it generates a random number and sends it back to the client . This number is then stored as the number associated with the client , for future reference .

Example provided :
Go to google
Log in
<img src='/images/welcome.gif'>
Then, based on ST 's off-site redirection policy, it changes any off-site (external) URLs to redirect through itself:
 Go to google
Log in
<img src='/images/welcome.gif'>
Then, based on ST 's off-site redirection policy, it changes any off-site (external) URLs to redirect through itself:
 Go to google
Log in
<img src='/images/welcome.gif'>
Next, it updates all on-site references to use the pseudonym. This makes all the URLs

unique:
 Go to google
Log in
<img src='/images/welcome.gif?38fa029f234fadc3 '>

Now because of what I have explained in the above paragraph when a phisher (or any other person / service / server / process) tries to match their list against the list from your cache , it would get a list of a lot of entrance pages and none of the internal pages would match owing to the random number appended at the end . One thing that we have to discuss here is the redirection policy that is going to be present in the server S . All of the content that you see on S is not really always hosted on S . It could have links to media content , pictures , documents that are hosted on another site . Additionally there could be pictures embedded on the server S that are hosted on another site . What is to be done then ? We can define these media contents to be of two types .In the first case knowledge of these media files can tell you about the site that you visited that is if a phisher knows you have media links to say pictures X.jpg , Y.jpg , Z.jpg and from this he can deduce for sure that you went to server S then these media files are deemed *unsafe* . On the other hand if the media files don't reveal anything about the server S on which they are linked to then they are labeled *safe* . For the latter we don't really have to do anything because as it is no information is revealed from these files .However, for the former we can do any of the two . Firstly if they are hosted on a sever B we could just send them to the client C and hope that things don't turn nasty . Secondly we could pass the media files through the *translator* of the server B. Here either B could use the random number generated from A , called shared redirection , or B could map the random number generated from A to another number which is known as transfer redirection .

Here the *translator* resides on the server side , the solution is independent of the user client . This is the ideal case where the user intervention is not really required . However , there are other ways that we could take care of the problem where the implementation is on the client side . To understand the second method to stop phishing or unwarranted information retrieval we first have to understand how data is stored from the web pages . A client C , when it goes to a server S , has some cookies / cache set from that server . Initially when things were started , browsers were developed the policy was going to be “same origin access” that is only the cookies set by the server maybe read by that server . Somewhere along the lines this was'nt really enforced . As a result now a phishing site could drop their cookie – which would be a script , that compares the list of all the sites that you have been to , to their list . And next time you access their site , they have what sites you went to . Cookies are used in a number of ways , one site may have a partner site that could read the cookie kept from the first site . Consider an example , we have a

site www.hulu.com that stores a lot of media content , on their own server . I make my site www.anexample.com that has media embedded from www.hulu.com . Now the user went and saw a clip on www.hulu.com . Next he/she comes to my site clicks on the same link .What happens now is that I read the cookie from hulu.com , see that the user has already seen the same clip that I am going to have to show to them and I set the user agent to read the file from their cache . This does save time for the user , saves network bandwidth as well . But it leaves a major vulnerability that can be easily manipulated to see the user footprints .

There are a number of ways that a user is legitimately tracked on the web . Single session tracking , multiple session tracking , cooperative tracking , semi cooperative tracking single session , semi cooperative multiple session – all self explanatory . What we want to do here is partition the cache . Like in any operating system , the hard drive is divided into various parts , each part assigned to a particular user , and a user can't read another's file . We are going to do the same thing from the point of view of the server . There are two things to be taken care of . First of all we need to disable third party cookie transaction . The third party cookie transaction can be defined as a cookie that is sent from a server S when the domain name specified in the cookie is not S . This is again not really something that you would like to have . It is a cookie from another domain being sent to the user cache , either S should state why it needs to get a cookie to be saved from another server or not save it at all . Secondly when ever a read request comes from the server side as to probing the status of the cache we make sure that the query is relevant . We only reply when the query pertains to the same domain that is strictly enforce the "same origin access policy" . Side by side we plug the *css* vulnerability by providing the functionality of visited site on the basis of cookie existence . In any case it makes sense to know that you visited a site if there was some user information generated for you from that site .

Let us now look at other attacks that are more technical and harder to plug . The first one of these is known as *SQL Injection* . This is pertinent to servers that allow the user to send a *query* to the server . Let us consider a client C interacting with the server S . He logs on to the server and is given the choice to enter a search query . The user then responds with typing a query that is submitted to the server . Now if the server does not properly validate the user input then an attacker can insert code into the query to make it logical still and be harmful . A simple snippet of code like "*or 1==1 --*" is more than enough to do the damage .It would always return with some result because of the 1==1 and the rest of the query after this snippet would be seen as a comment as "--" is seen a comment marker . The attacker then has access to a lot of "answers" that he needs . A simple query like that with the snippet can give an attacker all the user names and passwords , all that he needs to know .

Another type of attack is known as the *Cross Site Scripting* . This type of attack is more technical and pertains to the replies from the server . Searches on a particular site , normally any site return the same text string that has been entered as an input . They normally don't do a user validation at the point of the search input . They don't check whether the query in it makes sense or not . An attacker can do the following to get a user to download a javascript which would be used to read the users cookies / cache / key strokes . Let us see for example , a user enters in to the search panel the following "*<i>Hello World<i>*" . Here the server would return that *Hello World* not found . Note here that the server responds with the query text in italics because of the <i> . This is done because the search term as such received from the user is not checked for semantics or syntax . Consider what java scripts are . They are just text files which when coupled with the browser perform a particular task . In the same manner instead of a search term in the query an attacker could insert his own java code and redirect the result to the user . Here the user would see a blank screen and would in essence have downloaded the java script from the attacker from a totally different site . What should happen in such a case is that the user should be notified that the search term he entered was invalid and provide a link to get the same and if the user is dumb enough to actually not know that he is clicking on something that is definitely not his then in that case we can let him be . Another form of the same attack is the form redirecting XSS . In this type of attack the attacker would insert code in the form to redirect information to his own server / email before actually carrying on with whatever the user intends to do .

These forms of attack are very hard to "plug" . There are web vulnerability scanners like Nikto and Nessus that check if the server is doing something that it should not . They to the most parts are successful . However, because they are based on checking for the types of attacks based on previous occurrences they are utterly useless against instances which have no a priori history of happening . So we find another way to go about the problem. We define a generic class of problems that we need to look for . We don't really run anything on the client side . On the other hand we literally attack servers and find out if there is a vulnerability is , advise new sites coming up to scrutinize themselves using the same software . As explained above these types of security flaws basically exist due to ignorant web developers not thinking of all the scenarios and attackers taking advantage of the fact . Here we introduce a new application called *SECUBAT* - http://www.seclab.tuwien.ac.at/projects/secubat/ . Here is how it works . The application is divided into three parts – crawling , attacking , analysis . The first part of the application is akin to a web crawler , from a root site that the user specifies the crawler part spreads around , goes to all of the linked pages . Note here that we are more interested in forms that occur on pages . it is here that the user can input data .So after we know what all forms are there on what all pages , we commence with the next stage . We

are now ready to attack . For all of the types of vulnerability , the generic kinds we attack the sites . For the sql kinds by giving just ' , by just giving 1==1 , for the XSS kinds by giving java script snippets . After all the attacking is done , we analyze the data that we have generated . Remembering that we are the good guys, we inform the site about the vulnerability that is there so they can correct it . From the run that the inventors of Secubat they speculate that 6.33% of all the websites they went to , were vulnerable to Sql injection attacks . 4.33% for the simple XSS and 5.52% for the form redirecting XSS .

Most of the vulnerability that we see can be corrected . The last set of vulnerability are really easy to correct if the developer knows how to take care of them . We should also note that quiet a lot of softwares / features / applications that are built for the good can be used for nefarious purposes as well . As a user , your best defense is to have knowledge about how these things can be done . As they say knowledge is power . You can only take appropriate steps if you know what are the things that you need to be careful about . Also , there are a lot of people out there like our parents who are not that techno savvy , they click on anything that "appears" legitimate to them . We need for more solutions that plugs in the problems before it even presents itself to the user.

References:

1.R Dhamija, J.D Tygar, Marti Hearst "How Phishing works?", Conference on Human Factors in Computing Systems, pp581-590 ,2006

2.I.Fette, N.Sadeh,A.Tomasic "Learning to detect Phishing emails" , Proceedings of the 16th international conference on World Wide Web, pp 649-656 ,2007

3.D.Florencio, C.Herley "Stopping a Phishing attack Even when victims ignore the warnings" Microsoft Research

4.S.Kals,E.Kirda,C.Kruegel,N.Jovanovic,"SecuBat:A Web Vulnerability Scanner" Proceedings of the 15th international conference on World Wide Web, pp247-256 ,2006

5. C.Jackson , A.Bortz , D.Boneh , J.Mitchell "Protecting browser state from web privacy attacks" Proceedings of the 15th international conference on World Wide Web pp737-744, 2006

6.M.Jakobsson,A.Juels, J.Ratkiewicz, "Privacy-Preserving History Mining for Web Browsers"

7.M.Jakobsson, S.Stamm ,"Invasive Browser Sniffing and countermeasures", Proceedings of the 15th international conference on World Wide Web pp523-532,2006