Malicious URL Detection

Christophe Chong *[Stanford]*, Daniel Liu *[Stanford]*, and Wonhong Lee *[Neustar]*

Abstract—Web vulnerabilities are on the rise with the use of smartphones and mobile devices for both personal and professional use. This paper focuses on a machine learning solution that identifies malicious URLs using a combination of URL lexical features, JavaScript source features, and payload size. We use an SVM with a polynomial kernel to achieve an accuracy of 0.81 and an F1 score of 0.74.

Index Terms—Internet, security, machine learning, malware, javascript

1 Research goal

1.1 Mobile trends in security

In this paper, we discuss our experience training and testing a malicious URL detection system. Several trends in technology and security motivate our research efforts.

First, the web has grown to be an increasingly dangerous place. In 2011, Symantec reported a year-over-year increase in web attacks by 36% [5]. This translates into roughly 4,500 new attacks every day. The speed at which new attacks are deployed has considerably outpaced boxed anti-malware software.

Second, there has been significant growth in both personal and enterprise use of mobile web technology. In their 2012 State of Mobility Survey, Symantec noted that, Once mostly forbidden by IT, smartphones are now being used by hundreds of millions of employees throughout the world [6]. Thus, the attackable population for attackers has not only grown, but comprises, all else equal, a potentially more attractive group from a commercial or political standpoint.

Finally, although smartphone usage has grown to be a major endpoint in Internet connectivity, it still lacks the level of protection found in PC-based anti-malware technology. Part of this is due to the novelty of the devices, but it is also directly driven by strict controls some mobile device makers place on third-party software developers.

1.2 Research goal

Our ultimate goal is to contribute to the creation of a realtime malware classifier to block out malicious URLs. We focus our efforts on three subcategories of web attacks: drive-by-downloads, where users unwittingly download an executable malware payload; phishing, where attackers pose as legitimate websites to steal user information; and exploits from JavaScript code found in website source code.

2 Prior work

2.1 Lexical and URL features

We build upon the work of several malware detection research efforts. YoungHan Choi et al. use N-gram, Entropy, and Word Size as metrics for malicious code detection in JavaScript code [3]. They note that attackers commonly use obfuscation techniques to hide JavaScript attacks. Fortunately, the resulting code can often look heavily obfuscated in comparison to legitimate code, making it possible to use certain obfuscation patterns as a proxy for malicious intent.

Justin Ma et al. have demonstrated the potential of a classifier based on suspicious URLs [4]. They train their dataset on properties such as host-name length, overall URL length, and the count of the subdomain separating character (.). Combining these lexical features with host information (e.g. DNS registry info), the researchers report an accuracy rate of over 95%.

Finally, several researchers ([1], [2]) have used passive DNS query data to detect malware domains, particular those used to run command-and-control centers for infected machines.

3 Data

To find benign URLs, we seeded a crawler with the top 1000 websites on the Alexa ranking website and started collecting URLs. We then searched a malware database (http://support.clean-mx.de/clean-mx/viruses) to find recently discovered malware sites.

The problem with gathering data from malicious URLs is that, once known, the URLs dont stay up for very long. To save the state of the website, we downloaded the source or payload object at the URL location and saved it on a separate machine.

After beginning our analysis, we realized that lexical features were getting an incredible boost from duplicates. We thus removed URLs if their first three subdomains matched up. We ended up finding 701 duplicate URLs. At the end of the web crawls, we had 18K benign URLs and 14K malware URLs. We then randomly sampled18K malware URLs to get approximately 4500 benign URLs and 3500 malware URLs for our final training/testing set.

4 Features

We develop three different categories of features to detect malicious URLs.

4.1 URL lexical features

We approach the URL as an NLP problem. We use term frequency-inverse document frequency ($tf - idf$) to weigh the importance of a token in the URL as a way to associate URL tokens with labels. Tokens include anything in the URL, including both the domain and the path. $tf - idf$ can be defined as

$$tf \cdot idf = tf(t,d) \cdot idf(t,D)$$

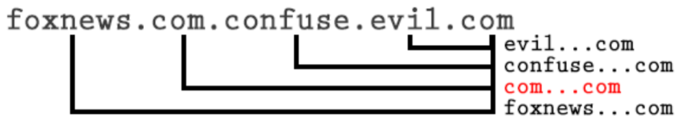
where we define tf and idf

$$tf(t,d) = \frac{f(t,d)}{\max\{f(w,d) : w \in d\}}$$

$$idf(t,D) = \log \frac{|D|}{|\{d \in D : t \in d\}|}$$

We also exploit the hierarchical nature of the subdomains by splitting along each separator and saving a bigram consisting of any subdomain plus the top level domain. We hope to run across phishing patterns or other suspicious URLs in the process.

Fig. 1: Attackers may systematically choose URLs



4.2 Source code features

JavaScript exploits are typically obfuscated to prevent detection by automated or manual analysis. Here's an example of one exploitative script we found in our malicious sample:

```
{k=i;s+=String["fro"+
"mCh"+"arCode"](n[k]/
(i-h*Math[f](i/h)+016));}
if(018-0xf===3)eval(s);}
```

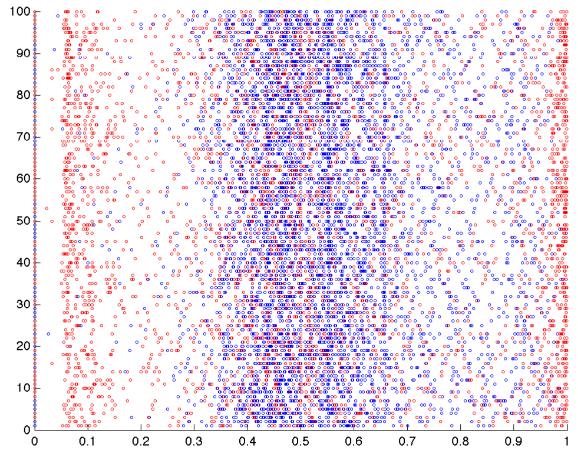
Fortunately, we are able to use the salience of obfuscation as a proxy for exploitative behavior. Attackers typically use special characters to encode script, either as direct ASCII or transformed by some simple character-to-character function:

```
document.write(unescape('
%3C%68%74%6D%6C%20
%6C%61%6E%67%3D%22
%65%6E%22%20%6...
```

Thus, we can use the ratio of special character subsequences (non-English for ”en” websites) to script length.

In addition, attackers who choose to reconstruct functions before calling them require the use of special functions, such

Fig. 2: Comparing special character subsequences in malicious (red) to benign (blue) URLs. The files are evenly distributed across the Y-axis for clarity. The X-axis is a normalized score.



as `fromCharCode`, `eval`, `document.write`, `escape`, etc. They can also include the malicious code in an `iframe`. We count these keywords and use them as one feature.

4.3 Network features

Although we have explored a variety of network features, including latency, DNS query data, domain registry data, and payload size, we have only captured payload size for our tests. Executable can be arbitrarily long, and obfuscated script may add to payload size as well.

5 Experiment

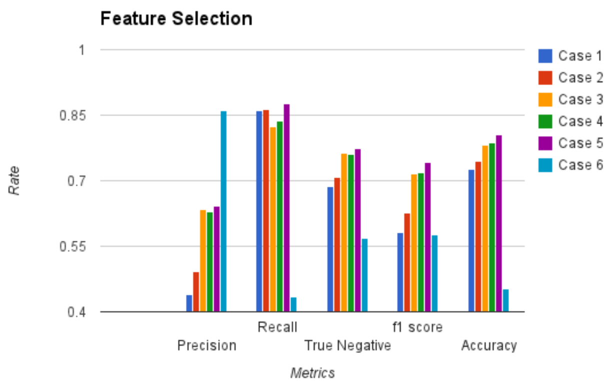
We chose to use an SVM with a polynomial kernel of degree two, with an 80/20 split on training and testing over 22K URLs.

We tried various combinations of the features: (1) lexical features; (2) lexical features and counts of keywords; (3) lexical features and the special character subsequence ratio; (4) lexical features, counts of keywords, special character ratio, and script length; (5) the previous case and file size; (6) the previous case without lexical features.

Our best results came from case (5), where all features are used. We had an accuracy rate of 0.81 and an F1 score of 0.74. Figure 3 summarizes our results for all cases.

Case	1	2	3	4	5	6
Lexical features	X	X	X	X	X	
Keyword count		X		X	X	X
Special char ratio			X	X	X	X
Script length				X	X	X
File size					X	X

Fig. 3: Malicious URL detection feature selection



6 Discussion

6.1 Results

One of our assumptions was that machine-generated malicious URLs could be detected by extracting lexical features from the URL. We have evidence to support that telling features can be extracted by using URL bigrams and tf-idf word association.

Although JavaScript features boosted our SVM performance, more sophisticated JavaScript features may be worth pursuing. One alternative includes extracting features by dynamically running and analyzing JavaScript code, which may end up having complementary performance with static JavaScript analysis. As we have observed already, legitimate URLs commonly engage in JavaScript obfuscation, making static code analysis a less attractive option.

6.2 Attacker strategy

The growing threat to mobile web users could be mitigated by automatic URL detection. By using a trained SVM, one could check URLs fast enough to deploy in a realtime service. This means users can use a preemptive service without impacting their mobile experience. As the old saying goes, an ounce of prevention is worth a pound of cure – but only if the solution is palatable.

Attackers may certainly make tradeoffs to outwit the features we have selected. However, such elusion isn't free. For example, using more legitimate sounding URLs in phishing attempts may bypass suspicious bigram detection, but may result in fewer click-throughs by scrupulous users. Or, reducing special char code sequences in obfuscation may work, but only by increasing script size or by using less obfuscation and risking detection by malicious code pattern detectors. Our hope is that by adding the appropriate features, a machine learning based system would be able to force attackers to make tradeoffs in web-based attacks.

7 Conclusion

We have shown that it is possible to construct an SVM to classify malicious URLs with some degree of accuracy. Future work would involve testing on a much wider array of malicious URLs, while incorporating a more sophisticated JavaScript feature extractor and utilizing more network features. More importantly, by using a trained SVM, it is possible to provide a realtime service to check malware URLs, regardless of the browsing device used.

In general, using a machine learning approach to discover malicious URLs and web attackers is a potentially significant approach, especially when considering the scale at which machines themselves have been used to automatically generate, obfuscate, or permute attacks. We hope to see more research put forward in this endeavor to further reduce the space of feasible attacks.

References

[1] Antonakakis, Manos. "Kopis: Detecting Malware Domains at the Upper DNS Hierarchy." http://static.usenix.org/events/sec11/tech/slides/antonakakis.pdf.

[2] Bilge, Leyla, Engin Kirda, Christopher Kruegel, Marco Balduzzi. "EXPOSURE: Finding Malicious Domains Using Passive DNS Analysis." http://www.syssec-project.eu/media/page-media/3/bilge-ndss11.pdf.

[3] Choi, YoungHan, TaeGhyoon Kim, SeokJin Choi . Automatic Detection for JavaScript Obfuscation Attacks in Web Pages through String Pattern Analysis. http://www.sersc.org/journals/IJSIA/vol4_no2_2010/2.pdf.

[4] Ma, Justin, Lawrence K. Saul, Stefan Savage, Geoffrey M. Voelker. "Beyond Blacklists: Learning to Detect Malicious Web Sites from Suspicious URLs." http://www.cs.berkeley.edu/~jtma/papers/beyondbl-kdd2009.pdf.

[5] Symantec. "Internet Security Threat Report, Volume 17." http://www.symantec.com/threatreport/.

[6] Symantec. "2012 State of Mobility Survey." http://www.symantec.com/about/news/resources/press_kits/detail.jsp?pkid=state-of-mobility-survey.