

# Web Client Attacks

Scribed by Gelareh Taban

April 21, 2008

## 1 Web Server Attacks – continued

We first conclude our discussion of detection of web server attacks from the previous lecture, which focused on the work of Kruegel and Vigna [KV03]. The authors introduce an anomaly detection system for web-based attacks and propose a number of features to identify anomalous behavior. Their model is particularly appealing as it is component based, where each component is independently trained. In the following, we briefly discuss some of the anomaly features.

**Attribute Length** We cannot assume a distribution on the attribute length, not even that it is a smooth curve. We also need to assume large variance as length can differ greatly based on function. Hence the most appropriate bound can be achieved using the Chebyshev inequality which is in general a very weak bound and only flags obvious outliers.

**Attribute Character Distribution** Exploits the idea that there exists a relationship between the characters in the query parameter. Is not based on the distribution of the actual characters, rather the pattern of the characters. This can identify attacks where for example, the attacker injects code in the function arguments and inserts unusual characters. As an example, the Code Red worm had a long sequence of the letter ‘A’ at the start of the code. The attribute character distribution by itself is not actionable—rather, it is used with the Pearson  $\chi^2$ -test to detect how likely the shape of the distribution of characters in an input has been seen before and therefore measure how anomalous the input is. (This is not a very intuitive parameter.)

**Structural Inference** This attribute models the structure of the query parameters using regular grammar. The idea is to learn the basic grammar of past queries and during the detection phase, determine if the new patterns match past patterns. Assuming that there are no negative pattern examples, we want to avoid *narrow* models (which exactly fit the training data) as they do not allow new patterns; and *overgeneralized* models (which fit all possible strings) as they allow *all* patterns. To generate a model, a Bayesian technique is used which assigns probabilities and conditional probabilities to all possible models and training data. The model probabilities are designed such that they favor simple models as they are more general (i.e., more data than just the training set fits the model). The idea is to generate lots of models, compute their “probability” of occurrence (simpler models are viewed as more probable), use training data on the model and finally decide on the best model. To detect, we use the model to see if there exists a probability that an input string will ever occur. If yes (even with a low probability), the string is ok. Else, the string is an anomaly.

Although the paper does a very good job of analyzing each of the features, it is weak in the following respects:

- They do not say what data they used (*i*) to develop their features; (*ii*) as the training set. In particular, although in their evaluation they found 11 exploits, we are not sure if the features were trained with knowledge of the exploits. Also why use 1,000 queries in the training phase? What happens if less or more are used? They could have explored the efficacy of different amounts of training queries to motivate the final choice.

## 2 Web Client Attacks

In web client attacks, a user can be attacked by simply visiting a malicious or infected URL. The attacker can detect vulnerabilities in the user’s web application and exploit the vulnerabilities by forcing the download of malware binaries. This type of attack can be modeled as pull-based infection—in contrast to web server attacks that use a push-based model [Provos07]. We review two important and similar initiatives to detect and track web client attacks, by Microsoft [Wang06] and Google [Provos07, Provos08].

## 2.1 Automated Web Patrol with Strider HoneyMonkeys [Wang06]

Unlike other approaches such as honeyfarms where the problems *come* to us, the work of Wang et al. from Microsoft Research, tries to *find* problems. The authors are the first to frame the eco-system that encompasses the malicious URLs and associated malware. Moreover, various intermediate results of the scheme are ‘actionable’—from the legal perspective to anti-spyware leads.

- The scheme runs a pipeline of “monkey programs” running possibly vulnerable browsers on virtual machines and tries to monitor and identify malicious web-sites. Monkey programs mimic a human user’s operation and therefore only consider automated sites. Their approach includes a *toxicology spread*: test a range of vulnerable things and see which are affected. In this case they test machines running different service packs and patch levels.
- The authors decouple the page and the badness and show the relationship through re-direction. Of course re-direction alone is not always bad since a lot of rich-text pages contain different levels of re-direction.
- The presented eco-system consists of a set of content-providers and exploit-providers. The content-providers feed the exploit-providers. This decoupling of the providers is particularly interesting from the market point of view. An interesting question to consider is to determine how many significant players there are in the system: a few, or thousands?
- The presented results show that once a patch comes out, there is a surge in attacks based on the vulnerability. This might reflect a use-it-or-lose-it time for the attackers, or instead might be due to attacker tools that auto-reverse-engineer exploits from patches.
- There is a lack of detail in the presented results: no assessment of false negatives. Also, although the authors mention their results are an upper bound, in some ways they are in fact a lower bound, as they are based only on Windows XP and a detection window of 2 minutes.

## 2.2 All Your iFRAMES Are Belong to Us [Prov07, Prov08]

A similar methodology was adopted by the Google people but at a much larger scale—the whole web. They crawled the web like the HoneyMoneys. They started with a few billion URLs and pre-filtered them to find the more interesting ones and ended up with around 4.5 million malicious URLs (as opposed to the few hundred of HoneyMonkeys). Once malicious URLs are discovered, Google labels the pages as potentially harmful and returns this label along with the website for search results (the URLs are still returned because the company views it as problematic to censor URLs). It is interesting that even though users are warned that a URL is potentially harmful multiple times through a series of re-directions, up to 20% of users still go through all the re-directions! (This fact comes from Vern’s recollection of Niel’s talk at HotBots. It does not appear in the paper.)

Unfortunately the authors [Prov07] don’t specify what range of browser configurations they used. In their later work [Prov08] the authors do not really tell us what heuristics they use.

The papers contain many very interesting graphs and tables. There are several interesting observations that can be made:

- Malware distribution networks are highly localized within common geographical bounds. A very large percentage of both malware and landing sites are hosted by China (next is the US). In fact, 96% of the landing sites point to malware distribution servers hosted in China. (See Tables 2 and 3 in [Prov08]). However, Vern speculates that this *might* be due to the use by the attackers of domain registrars based out of China, with the actual sites being elsewhere.
- An interesting question is “where are the badness?”. Adult sites have a stronger rate of hosting badness, but in sheer numbers they do *not* dominate web-based attack sites. Figure 4(b) shows that when all 3.3M malicious URLs were categorized, adult web-sites are much lower fraction of total landing sites. So even if you avoid adult web-sites, you remain exposed to risk!

## 3 References

[KV03 ] Christopher Kruegel and Giovanni Vigna, *Anomaly Detection of Web-based Attacks*, CCS 2003.

- [**Wang06**] Yi-Min Wang, Doug Beck, Xuxian Jiang, Roussi Roussev, Chad Verbowski, Shuo Chen, and Sam King, *Automated Web Patrol with Strider HoneyMonkeys: Finding Web Sites That Exploit Browser Vulnerabilities*, NDSS 2006
- [**Provos07**] Niels Provos, Dean McNamee, Panayiotis Mavrommatis, Ke Wang and Nagendra Modadugu, *The Ghost in the Browser: Analysis of Web-based Malware*, HotBots 2007
- [**Provos08**] Niels Provos, Panayiotis Mavrommatis, Moheeb Abu Rajab and Fabian Monrose, *All Your iFRAMEs Point to Us*, Google technical report provos-2008a. <http://research.google.com/archive/provos-2008a.pdf>