Security Solution for Company X

An Anti-Web-Crawling System
Based on Tomcat Server Log
And
Security Proposal

Linxiao Bai
Master of Science
Gorgen Institute for Data Science
University of Rochester
lbai2@ur.rochester.edu

1. Introductions

To protect the privacy of our sponsor. The name of the sponsoring company is replaced with a fake name X.

X is a clinical decision support system (CDSS) software intended to be used by medical practitioners, including primary care practitioners, to assist them in differential diagnosis [1]. It is a knowledge database providing answers to clinical queries. Like any knowledge database, the most precious intellectual property of X is its stored information. Particularly, the data bases for clinical image and diagnosis are the foundation of this company.

Unfortunately, abuses of computer technologies make massive database stealing possible. In an identified hacking event in 2015, a hacker stole thousands of images in a matter of hours. This is a serious issue because hacking behaviors not only threaten the intellectual property of the company, but also jam online traffic and compromise user experience.

Our sponsor X asks our team to design strategies that help with the security of their system. This paper presents the solution we give X in response to their security problems.

The first part of the paper is the result of the exploratory analysis. It reports several security breaches we found in the system. It also provides industrial-level solutions adopted by other companies to fixing similar breaches.

The second part of the paper will focus on a data-driven algorithm we designed for identifying hackers. Suggestions to online implementation of the algorithm and execution strategies are also provided to the sponsor.

Finally, the paper provides perspective to the future work of the project.

2. System Evaluations and Found Security Breaches.

Exploratory analysis of the current X system was carried out carefully to explore the relationship between visitor actions and system responses. In the manner of aggressive visiting, mimic attacking actions are applied to the system to test its response. For example, session ID reacquiring by repeated login-logout, recursive image acquiring by url assembly, and robotic http visits by forging request fields.

During the exploratory tests of the system, some security breaches are found, and worth immediate attention.

Lack of HotLinking Protection.

One of the most effective anti-crawling (especially for images) measures currently applied by the industry is HotLinking protection. This protection forbids random resource visits without valid source urls (referrer).

Most web-crawling relies on url assembly to collect resources from the server. Usually this type of visit does not carry source information because the url is assembled by code, and the visit is direct to target ignoring web-browsing consistency.

Experiments show that both X6 and X7 do not execute referrer check for picture url. The image url of a diagnosis search is copied from one machine and pasted to the browser of another machine. The image is returned without any problem. In another test, image id field of the url is changed at random fashion instead of required from diagnosis searches. Result shows images are returned as long as the image ID is valid.

This is a dangerous signal. Attackers can easily gather massive number of pictures in a short time by recursively replacing image ID.

Redemption:

First, all current sessions contain image request but without any referrer needs attention. Those actions could be resulted from direct visit of image url, which are anomalies.

Second, one protection X should immediately implement is referrer binding. That is binding each image url with possible referrer (url of searches and diagnosis), and suppressing image requests whose referrer does not match the white list. This strategy requires a back search of the image against the diagnosis or symptoms. The referrer check does not need to be real-time and strictly before returning results. A parallel progress can be executed at a delayed-real-time manner to check the white list and take actions against illegal sessions. This will guarantee a fast search user experience.

Unrestricted Login Interface.

The login interface of the X lacks robot checking procedures. Experiments show that after 10 consecutive login-logoff actions, the system does not take actions against robotic login. This gives attackers unlimited easy accesses to new session ids. The immediate result of this breach is it challenges any analysis based on session ID. Because attackers can logoff immediately after a mild attack and login with a new session ID. Also, if X were to construct a real-time intrusion analyzing platform in the future, it must rely on the credibility of session ID.

Redemption:

The most effective and simplest fix is to enforce robot checking on users with frequent login and logoff actions. For example, CAPTCHA. Computer vision techniques may penetrate such protection, but our goal is to make intrusion expensive.

Do Not Enforce Login on Industrial Users.

The current X authentication system gives permission to all users whose request comes from subscribed IP. For those users, their actions are only being recorded by session ID. Their blurry identity makes tracking responsibility hard. Also, it causes trouble to any analysis based on user level.

Redemption:

It is possible to provide unlimited services to industrial users under certain IPs and ask for a logged-in account at the same time. Also, extend the session length of those users should help with their user experience.

Unenforced Indigenous Agents Check.

The nature of mobile app determines their visits are safe and mild. Also, almost 70% of the requests come from indigenous apps of X. So, it is highly necessary to build a special barrier for indigenous agents such that the system can trust the credibility of request coming from those agents. Also, the burden of intrusion detection will be greatly reduced.

Based on the information from the database, it appears that the apps of X forward the app information unhashed. Also, during each request, the authenticity of app remains unchecked. The direct result is that attackers can easily fake the authenticity of the indigenous app and trick our presumed trust on indigenous products.

Redemption:

It is recommended that all indigenous products use hashed IDs to declare their identity to the server. Meanwhile a white list check is enforced for all requests declaring indigenous products' identity. In this way, attackers are unable to obtain the indigenous product's identity without decompiling the installer and use it to bypass the white list. By doing this, not only a reliable source of trusted users can be safely labeled and stay unharmed, but also a positive sample of normal users can be quickly collected for further analysis.

3. Data Driven Intrusion Detection Algorithm.

A robust and secure system will greatly reduce its chance of being attacked. However, it is almost impossible to eliminate hacking behavior because hackers will always have the initiatives. This is because system protection strategies are always passive and dependent on known or imaginary hacking behaviors. Also, hackers can adjust their strategies correspondingly and take counter measures against the system fire-wall. For example, robot checking based on lettered CAPTCHA is no longer considered safe in recent years because of the abuse of computer vision techniques.

On the other hand, it is possible to construct a data driven model to analyze the behavior of suspects' actions and identify hacking behavior based on anomalies. This protection will not only suppress aggressive behaviors against the system, but also collect evidence of hacking for further legal actions.

Given the current condition of the X system, our objective is to design an algorithm that can identify happened intrusions. Fortunately, X keeps its visitor logging complete making model building possible and easy.

Data Summary.

Two data source are used in our analysis, both are from tomcat server log, event and apache_archive. Event data keeps track of user action and other associated information triggered by java scripts. While the apache_archive keeps track of all information returned from the server in response of user actions. Both data sources have very high qualities, no data cleaning are performed.

Based on the condition of X system, data sub-setting and assumptions are applied to generate interested universe for analysis:

- Split users by agent type (mobile user, non-mobile user).
- Trust indigenous app for its innocuous traffic.
- Only focus on relative recent sessions, sessions in the January of 2017 is selected.
- Exclude industrial users, even if intrusion was found, finding responsible individual is impossible.
- Use session ID as object identifier, group actions by session ID, order by time in each group.

The result data reduces the server log to the form that each object is identified by session ID and its exact actions sequence of visit is represented as a list.

Evaluation of Action Sequence Rareness

Our team makes a major assumption about user behavior. That is the next action of a user during a visit is dependent on its previous actions. For example, users who did search action are naturally likely to do a view image action. The objective is to construct a conditional probability distribution of user's next action given its previous action. Figure 1 shows the probability model used to describe actions in a session.

$$
\begin{aligned}
Start &\rightarrow P(A_1) \\
&\rightarrow P(A_2|A_1) \\
&\rightarrow P(A_3|A_2, A_1) \\
&\cdot \\
&\cdot \\
&\cdot \\
&\rightarrow P(A_n|A_{n-1}, A_{n-2}, \cdots, A_1) \\
&\rightarrow End
\end{aligned}
$$

Figure 1. A probability model describing user actions

The corresponding probability can be learned from the data depending on different choices of probability models. Common choices include Bayesian Network, Recurrent Neural

Network LSTM, and Hidden Markov Model. Among these three choices, our team discover that RNN gives the best result.

Between each pair of actions, RNN model evaluates its predicted next action against the ground truth and yield a action-transition loss using Log-Softmax function [2]. The loss is a representation of how the actual action deviates from our expectation based on evaluation of the data. Also, because the data represents a behavior of the general public, the loss will be low on the majority actions and high on anomalies.

Average loss of session is then computed to give a rareness score of the session. Notice that the average loss is independent of the number of actions in that sessions. Compared to traditional intrusion detection based on volume, it is a brand-new dimension that provides more decision space.

Figure 2 display the rareness distribution of non-mobile users (top 500), where sessions are ranked by its score. Right part of the curve represents high rareness and thus high ranking. Also, an identified hacker is marked by a red dot in the plot to visualize the performance of rareness score.

A decision boundary can then be drawn to determine a critical value. We recommend a threshold determine strategy based on the score differences of neighboring ranking sessions.

We define $\Delta S_i = \frac{S_{i-1} - S_i}{S_i}$. Where S_i and S_{i-1} is the rareness score of i-th rank and the one above. The mathematical meaning of ΔS_i is simply the changing rate of score regarding its rank position. Plot of i against ΔS_i can then be drawn. The result is shown in Figure 3. We arbitrarily set a threshold to be around 30th rank with rareness score of 2.167 based on the fluctuation of the curve. Any session whose rareness score is beyond 2.167 is deemed rare. The reason is after 30th rank, the fluctuation of rareness changing rate increases dramatically.

The toy reasoning behind this threshold cutting method is the distribution of rareness score is a long-tailed distribution [3], and the point where fluctuation happen is the point where the observations breaks the smooth increment trend and experience the long-tail effect. In this case, the long-tail effect happens in around 30th rank.

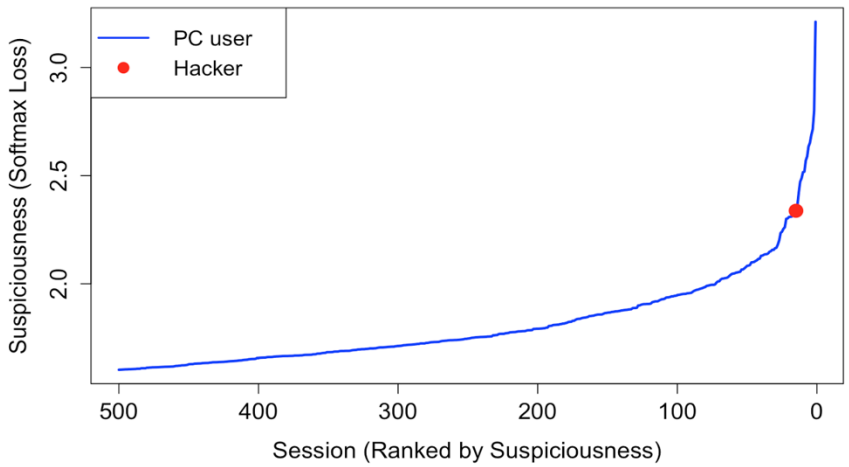

Figure 2. Session rareness ranking.

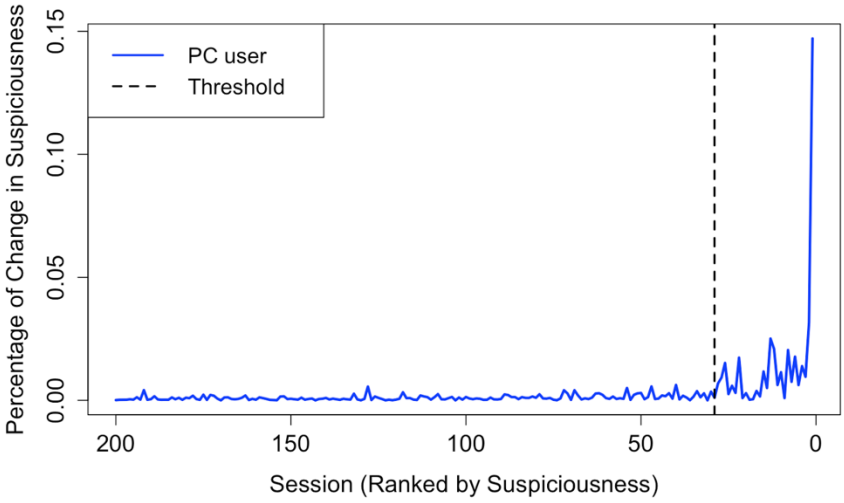

Figure 3. Changing rate of rareness, and threshold setting.

The fact that hacker session appears in the top-20 rareness is a strong support of the effectiveness of the rareness score. In fact, the hacker's massive abuse of image-getting-action without variety earns its high score. Furthermore, we manually inspect some of the high rareness sessions. All of them suffer from infrequent actions that are not honored by normal users.

For example, the top-1 session is extremely rare because it frequently uses the "quiz history" function of the system. Similar situation happens to other sessions too. Their visits to minority links result in the high rareness. Sessions like these are infrequent but totally legal and innocuous. However, our concern limits to aggressive and dangerous actions such as frequent image downloads and resources visits. The support of another dimension is needed to determine the aggressiveness of such visits.

Ideally, we wish to find a joint decision-making strategy that labels sessions not only by rareness, but also by viciousness.

Analysis of Image Request Volume

The biggest threat of X is image stealing. In one session of the identified hack, more than 7000 images were stolen from the server. Naturally, the amount of image request in a session is a perfect indicator of its potential viciousness.

It is worth noticing that volume based analysis by itself should not be considered as a determinant factor for intrusion identification. This is because from this dimension along, heavy users and frequent users are easily misclassified as vicious, while the true hackers will try everything to avoid being obvious on the volume chart because they know it is the first-place people check for intrusion.

However, with the rareness score proposed, it is safe to use image request volume as an auxiliary dimension for viciousness evaluation. Similar as our previous strategies, we treat request count as a score for potential viciousness. Figure 4 shows the distribution of session request count with respect to its image count ranking in the population.

Figure 4 also shows that image count is the dominant type of request of all because the dotted curve occupied around 0.7 of the area under the solid curve for both platforms. Also, image request has a strong correlation with the total request. This is proved by the similar trend of the solid and dotted curves. Finally, the request-count distributions of different platforms are very similar. This finding is very surprising because we thought uses on mobile platforms are more light-weighted. However, figure 4 shows that the needs are all the same regardless of the platform.

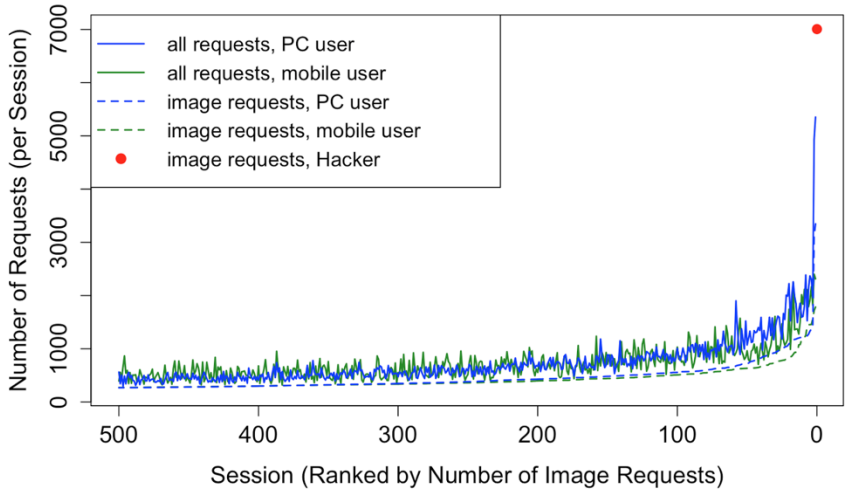

Figure 4. Request-count distribution, non-mobile represented as PC

Particularly for non-mobile users, we are concerned with setting a critical value for determining vicious sessions that download massive images. Similar as before we define: $\Delta V_i = \frac{V_{i-1} - V_i}{V_i}$.

Where V_i and V_{i-1} represents the volume of corresponding rank. The mathematical meaning of ΔV_i is the chaning rate of volume at its rank position. Figure 5 displays the the distribution of ΔV_i with respect to its rank i. Similarly, a threshold of 4th rank at 1,243 image requests in one session is determined based on the flucturaion.

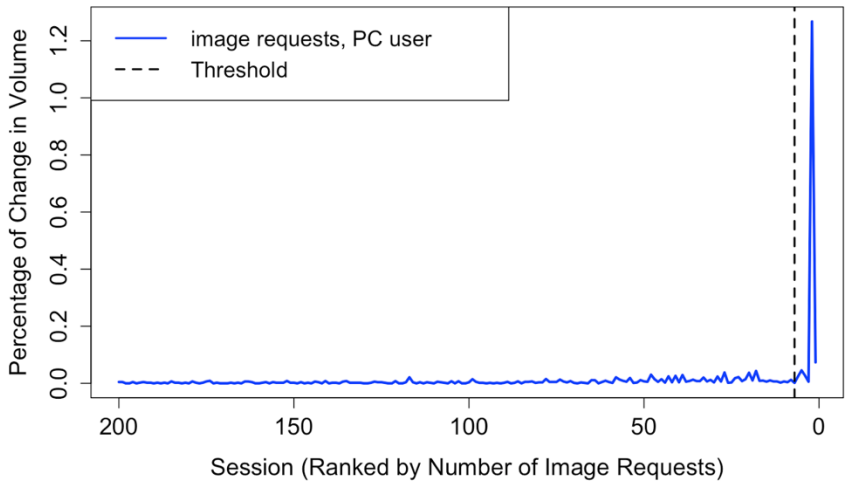

Figure 5. Image count change rate and threshold setting.

Joint Decision Space and Decision Region

Two analysis we have done so far evaluate session from two independent dimensions, rareness and viciousness. Together, they form a 2-D decision space where a session must be rare and carry vicious potential at the same time to trigger the alarm. Figure 6 shows a scatter plot of all sessions and their position in the decision space we design.

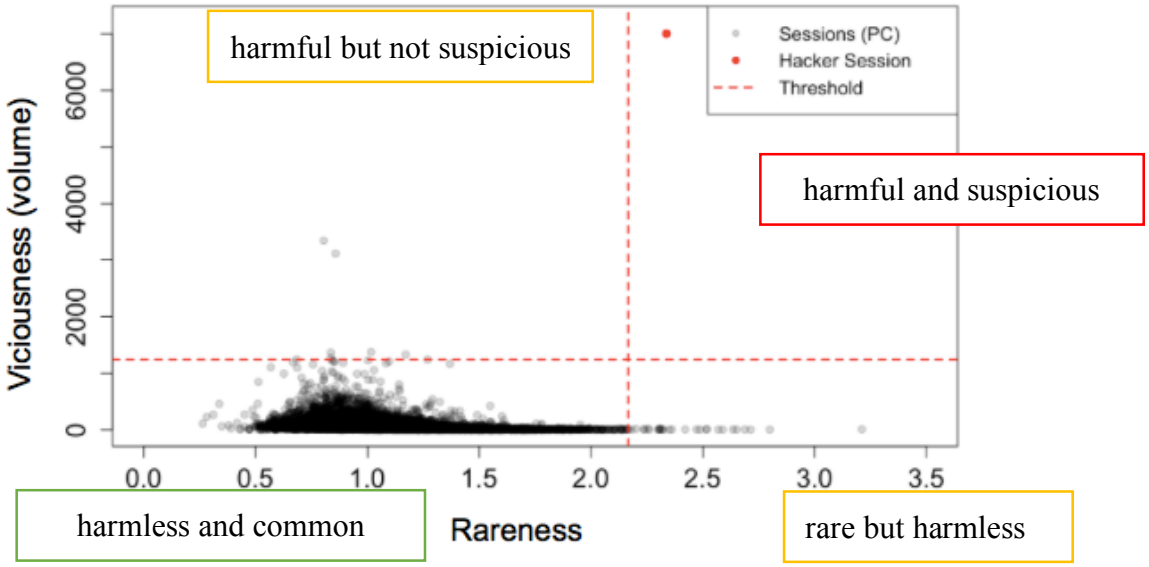

Figure 6. Sessions scatter plot. Upper right region is the critical region.

The first observation in figure 6 is the identified hacker is located at the critical region. Notice that the process of making the decision region is independent of the knowledge of the known hacker. The system self-identified the hacker.

Second, most sessions are located at bottom-left corner, where sessions are both common and harmless.

Third, sessions that crosses one boundary but not the other are of minority. These sessions can be interpreted as legal rare events. For example, a user is either continuous using the system for too long that cross the volume boundary or participate in infrequent actions that cross the rareness boundary. Either case, those sessions are far away from the other boundary and is not identified as hackers.

Online Strategies and Implementation Suggestions.

Except from the known breaches that need fixing. The data driven detection algorithm we design are also of real-time computation ability. The rareness score can be dynamically programed such that the loss of the next action can be computed within constant time. The additive image-request count can also be computed in real-time. A parallel progress that monitors and evaluates all live sessions should do the work.

Furthermore, we propose a three-strike protocol as the online execution strategy, where each strike is defined as the same user crossing the critical region during a period of time, for example, one week.

- Strike One: Terminate the session immediately.
- Strike Two: Terminate the session and suspend the account for a short period, and alert X personnel.
- Strike Three: Terminate the session and freeze the account until manual inspection.

In our strategy, sessions are terminated immediately if crossing the critical region. This will protect the server from immediate harm. Also, with the help of robot checking at login interface, a Turing test is automatically enforced. The second and the third strike are further protection of the system. They enforce stricter restriction on the intrusion suspects, and they also require the participation of the personnel for safety precaution.

4. Conclusion

The first part of the report evaluates the current system of X. Several security breaches are reported for fixing. Also, a future perspective of a security system is provided. It outlined an online real-time platform focusing on rule-based white lists and data driven algorithms.

The second part of the report focuses on an intrusion detection algorithm designed by us. The algorithm evaluates a session in two independent dimensions, rareness and viciousness. A joint decision space is then constructed based on independent threshold cutting. The formal definition of a hacker by our algorithm is a session that falls in the critical region.

5. Future work

Restricted by the development time and sql-query cost, our algorithm only runs on one month's sessions in apache_archive data and excluded industrial users. Further work may be conducted based on industrial users in longer time period to check for intrusions.

Also, a real-time streaming version of the algorithm can be implemented using Java and apache servlet. This will become an extreme powerful tool the industries eager to solve web-crawling intrusions.

6. References

[1] X. (2017, April 27). Retrieved May 04, 2017, from https://en.wikipedia.org/wiki/

[2] Bishop, Christopher M. (2006). *Pattern Recognition and Machine Learning*. Springer.

[3] Asmussen, S. R. (2003). "Steady-State Properties of GI/G/1". Applied Probability and Queues. Stochastic Modelling and Applied Probability. 51. pp. 266–301. doi:10.1007/0-387-21525-5_10. ISBN 978-0-387-00211-8.

[4] von Ahn, Luis; Blum, Manuel; Hopper, Nicholas J.; Langford, John (May 2003). CAPTCHA: Using Hard AI Problems for Security. EUROCRYPT 2003: International Conference on the Theory and Applications of Cryptographic Techniques.

[5] Thomas Scott (2004-07-13). "Smarter Image Hotlinking Prevention". alistapart.com. Retrieved 2007-11-16.