PalProtect: A Collaborative Security Approach to Comment Spam

Benny Wong, Michael E. Locasto, Angelos D. Keromytis

March 22, 2006

Abstract

Collaborative security is a promising solution to many types of security problems. Organizations and individuals often have a limited amount of resources to detect and respond to the threat of automated attacks. Enabling them to take advantage of the resources of their peers by sharing information related to such threats is a major step towards automating defense systems.

In particular, comment spam posted on blogs as a way for attackers to do Search Engine Optimization (SEO) is a major annoyance. Many measures have been proposed to thwart such spam, but all such measures are currently enacted and operate within one administrative domain. We propose and implement a system for cross-domain information sharing to improve the quality and speed of defense against such spam.

1 Introduction

The use of blogging software is an extremely popular web application. Blogging complements traditional means of Internet communication like email and instant messaging. The use of blogs ranges from personal journals to corporate marketing. More serious blogs attempt to supplement or replace traditional journalism and political punditry.

While blogs facilitate group discussion, it is the very interactive nature of this communication that opens it to attack. Often, the blog software and users themselves are not the target of attack. Rather, the application is used as a platform for Search Engine Optimization. Attackers use the comment posting functionality of blogs to submit comments that include links to the sites they want to advertise. When search engines index the blog, including the comments of each post, the attacker hopes that the sheer number of links that he has inserted across a number of blogs will increase the ranking of the target site in search results.

Besides threatening to decrease the quality of search results, spam comments waste the time and resources of blog owners and hosting companies. The insertion of comment spam is often automated, and the sheer amount of it makes manually identifying and deleting comment spam a time–consuming process, in addition to taking up a significant portion of the space that a blog owner may have allocated (or been allocated, in a hosted environment) for the blog content.

Disabling comments is a somewhat unsatisfactory solution to this problem. It is akin to an email spam solution where no-one is allowed to send an unsolicited email (and thus no-one can start an email conversation with an unknown contact). Such strategies are self-defeating. What is needed is a high-quality, high-confidence, automated mechanism for identifying and deleting comment spam as it is submitted. While a number of technologies have been proposed and implemented (mostly adapted from current email spam solutions), none perform automated cross-domain sharing of comment spam signatures.

We present the design and implementation of PalProtect, a plug-in for the popular WordPress blogging software. PalProtect automatically identifies comment spam, creates a signature for it, and distributes the signature to a collection of peers. PalProtect is a concrete example of a collaborative security system.

2 Related Work

Collaborative security is the growing trend towards sharing information security resources within and across administrative domains and systems to improve the overall security of the peer group. Three areas of computer security where a collaborative approach are immediately applicable are (*a*) worm detection and notification, (*b*) self-healing software, and (*c*) spam filtering. The reasoning is that a larger and more widespread network of sensors can achieve more complete knowledge of an attack more quickly than a single isolated node.

This observation is a widespread one. In particular, for worm detection [1], notification [2], and containment [3] systems, a collaborative approach is mentioned several times in the literature. Systems that seek to generate signatures for worm traffic include Autograph [4] and EarlyBird [5]. Both papers refer to signature distribution as a fundamental step in defending against worms. A recent study by Moore *et al.* [2] concludes that a worm containment response needs to occur within three minutes. In addition, the participation of nearly all major AS's is required for a containment to be effective. While these requirements are quite challenging, they confirm that foreseeable threats are best addressed by a collaborative approach.

Vigilante [6] is a system motivated by the need to contain Internet worms. To that end, Vigilante supplies a mechanism to detect an exploited vulnerability. A major advantage of this vulnerability-specific approach is that Vigilante is exploit-agnostic and can potentially be used to defend against polymorphic worms. While Vigilante doesn't address the self-healing of a piece of exploited software, it defines an architecture for production and verification of Self-Certifying Alerts (SCA's), a data structure for exchanging information about the discovered vulnerability. Vigilante works by analyzing the control flow path taken by executing injected code.

Collaborative security can also be leveraged for more mundane intrusion detection tasks. DOMINO [7] is a system for correlating intrusion alerts. Lincoln *et al.* examine the problem of privacy–preserving alert sharing for IDS systems [8], one of the challenges proposed in Du and Atallah [9]. Kruegel *et al.* [10] propose a peer-to-peer system that recognizes attacks in a distributed manner. In their system, only a small number of messages needed to be exchanged to determine that an attack was underway.

A collaborative approach to security also seems useful in the context of self-healing software. Not only can networks and end-hosts exchange information about intrusion alerts, but they can also exchange information about exploited vulnerabilities and code patches for these vulnerabilities. Application Communities [11] are one particular expression of this idea whereby a large collection of hosts agree to collaboratively monitor small slices of each instance of an application locally. When a fault or vulnerability is discovered, information that enables the host to prevent further occurrences of that fault is exchanged with peers.

The system most closely related to ours is Vipul's Razor (`http://razor.sourceforge.net`). It is a reputation-based system for filtering email spam, but the identification of spam is not automated. Like most collaborative security approaches, it requires some amount of community buy-in to increase its effectiveness. Challenges for this system (as with ours) also include privacy-preservation, trust and reputation issues, and peers with differing profiles. While some of these problems can be solved with white-listing, our approach uses Z-strings to help address privacy concerns. Z-strings are one-way data structures that remove the ability to reconstruct the original input that forms a signature, but can still be used to match.

3 Architecture

PalProtect is a plug-in to the WordPress blog software that enables weblog owners to prevent spam comments from entering the system. It does this by creating signatures of comments that have been identified as spam. Comments can be classified as spam by both automated (another anti-spam plug-in) or manual (blog administrator review) mechanisms. This information can then be used to identify future comments that may be spam.

In addition, PalProtect not only uses this information to prevent potential spam comments, it shares this information with its peers so that they can benefit from it as well. When a signature is created for the blog that initially catches the spam comment, and the signature is inserted into PalProtect's database, PalProtect proceeds to notify its peers of the new signature.

3.1 Spam Detection

Initial classification of a comment as spam can be done through a number of other anti-spam plug-ins. We treat these plug-ins as sensors to PalProtect. PalProtect's primary function is correlation, although it does maintain its own signature database as a last resort to classify and block comment spam. These sensors include Akismet, Bayesian Comment Spam filter, Spam Assassin, WP Blacklist, and Graphic Turing Tests (GTT's). When one of these filters classifies a comment as a suspected piece of spam, PalProtect will be invoked.

In order to reduce dependencies on other modules and plug-ins, one major assumption of PalProtect's design is that PalProtect can observe some external action, signal, or notification that other anti-spam plug-ins exhibit. Provided that such a signal exists, PalProtect captures the raw text data of the comment. From there, it creates a signature from the spam comment and saves it to the WordPress database. After this information has been saved, the raw comment is forwarded to peers. The methods of signature creation and methods of broadcasting are discussed in Section 3.3 and Section 3.4.

3.2 Enforcement

In addition to providing its own enforcement via matching against its internal signature database, PalProtect still leverages any other spam filters that are present in the blog. In this way, PalProtect is positioned as a last resort mechanism to catch comments suspected of being spam. If the comment has passed all of the other filters that are in place, PalProtect compares the contents of the comment with the signatures in its database. The comment can only enter the system once it has passed through all of the tests, including PalProtect's enforcement.

PalProtect can employ a variety of signature types to match and discard spam comments. Depending on the type of signature preference that is currently set, PalProtect converts the new comment into an instance of the current signature type. From there, PalProtect will use signature specific methods to determine whether the comment is in fact recognizable as a piece of comment spam. For example, using URL lists (discussed in Section 3.3) PalProtect will flag comments that have half of the URLs that are in a URL list. However, for the hash function, matching is based on taking a hash of the raw comment text and comparing it with the hash values already in the signature database.

3.3 Signature Creation

The data in the comment can be modeled in a variety of ways to create signatures for matching spam. PalProtect provides five ways to create these signatures. User-defined signature creation methods are easy to integrate into PalProtect. Because of this, the range of signature creation methods is easily extensible. The five base methods that are integrated into PalProtect are:

1. **Exact Match** - This method takes the raw data from the comment received and compares it with the raw data that is in the database. This is the simplest and quickest, but arguably the least effective at catching even slightly polymorphic spam.

2. **Longest Common Substring** - This method is similar to the exact match, except that it will take substrings of the data in the database and compare that to substrings in the pending comment. In the database, it will not be stored in an extra table because the data used is stored for the exact match.

3. **URL Lists** - Since the purpose of most spam comments is to lead the reader or a search engine to another site, one of the most effective ways to identify spam threats is to extract the URLs from the spam comments. PalProtect will extract all of the URLs in the spam comment and store it as an array in the database. The way URLs are stored can be full URLs or a smaller substring (i.e. the domain name) to broaden the scope of enforcement.
4. **Hashing** - Another way to create these signatures is by hashing the data and storing the hash key into the database. This is used mostly with the comment data, but there is a possibility that it can be used on URLs.
5. **Z-String** - Using the Z-String method, we can create a string signature based on the frequency of the letters in the comment. This method is effective when the spam comments are different, but similar enough that most of the comment is the same. The secondary purpose of the Z-string is to assist user privacy. The original message cannot be recovered, but the Z-string is still the basis of a good signature. Of course, this this case, peers would not forward the raw text, but rather the pre-computed Z-string.

3.4 Message Packaging and Encoding

The main motivation behind creating this plug-in is to record information that we have learned and notify our "peers" of this information. The method that is used to encapsulate and distribute that data is a crucial part of the process. For each comment broadcast, the data will be sent to each peer along with the URL of the source blog. This information will serve to identify the sender. This is important to include because we want to avoid sending the message back to its sender.

Each message will also be PGP-encoded to ensure that the message sent was indeed a valid message. Each PalProtect installation will have a buddy list to hold all of its peers information (e.g., URLs and PGP-key information). This list will be discussed further in Section 3.6.

Encoding the message with a PGP-key will cut down significantly on user intervention as well as ensure that the message is an authentic message from a peer. Instead of having a message enter a queue or an "inbox" of some sort where the user of the receiving blog must approve the signature, the PGP-encoded message can be automatically approved provided that it is signed with a valid peer public key and encrypted with the receiver's public key. If decryption fails, the message was not meant for this PalProtect instance. If verification fails, the message was forged. Extending and delegating trust (perhaps via a system like KeyNote) is interesting future work.

3.5 Sending and Receiving

This package of information will be sent to the peer using an HTTP POST request that is generated by the PHP code. The POST request contains these two parameters: the URL of the sender and the PGP-encoded message. Since we are sending simple text between the blogs, a POST request is the perfect vehicle and a low-overhead method.

The way PalProtect receives the messages from its peers is by having a dedicated page for it to receive messages. Since it will never have to display anything, it is a page containing only PHP and will parse the POST parameters that it receives. Once the message is received and decoded, the raw comment is first checked against PalProtect's local database to make sure that it is not a duplicate signature. If the comment is new, then it is inserted into the local database (and optionally forwarded to a set of peers).

3.6 Buddy List

PalProtect uses an array of objects to keep track of its "buddies." The objects will contain two pieces of information: the URL of the peer blog and the PGP-key of that blog. To ensure the integrity of the information in the buddy list, the URL and PGP-key will have to be entered by the owner of the blog. This will prevent problems that may arise from any automated system.

3.7 Additional Functionality

A previously undiscussed feature of PalProtect is the ability to maintain a white-list. This white-list provides the capability for the user to guard against having certain messages enter the signature list. In particular, attackers may attempt to submit spam comments that include links to popular legitimate sites like `cnn.com`, `citibank.com`, and `whitehouse.gov` or other strings that the user deems acceptable. Without a white-list, We require the new table due to some limitations of the list management functionality in WordPress's API. For every comment that is considered spam, PalProtect creates an encapsulated object that will hold the raw text as well as any of the other signature representations determined by PalProtect's settings. By creating a table specifically for the signatures, each record will represent one comment with the columns being the different signature representations. This organization makes the signature creation scalable by easily adding a column if needed.

When PalProtect receives a comment, it will first check the signature creation method preference that is currently set. It immediately stores the raw text in the object and then proceeds to store the newly created signature.

3.8 Limitations

Many of the problems that PalProtect may encounter are problems with most intrusion detection systems. One of the more common problems is that of identifying comments as "false positives." If there is a comment that is wrongly identified as spam and has had a signature created and distributed, we need to find a way to remove this entry from the list of signatures and somehow relay that message to the peers that we have sent it to. This problem is partially addressed by creating a revocation message type, but some process that identifies false positives still needs to be established.

One sensor that we would like specifically exploit would be the Graphic Turing Test (or GTT) due to its high confidence system to detect whether the commenter is human or not. The basis of the Graphic Turing Test is a challenge-response system. Given an image of distorted alphanumeric characters, a human will be able to distinguish the characters while an automated process will find it difficult. If the GTT field is left empty or is incorrectly guessed, this is a strong indication that the comment was posted by an automated process and should be considered spam.

However, we are unable to utilize the the GTT sensor. The other sensors that we employed to catch spam would only mark a comment appropriately. However, when the GTT refuses a comment, it automatically prevents the comment data from entering the database. Because of this, the comment would not be able to be marked as spam – thus depriving us of the event that PalProtect uses. The GTT directly manipulates the database in its code (which does not involve WordPress itself), thus eliminating the medium that PalProtect and GTT would potentially communicate through. Since one of our design requirements was to be standalone and have no dependencies on or changes made to other modules, we are currently unable to use the GTT plug-in as a sensor.

Another problem that arises in our system is in the distribution of the spam data. Currently, PalProtect iterates through the buddy list and broadcasts a comment out to everyone on the list, except to the immediate sender. We would ideally want the system to forward the message to a few peers, and then have them forward it to others – in essence, controlled flooding. This approach would be the most effective way to distribute messages over a large network and would also alleviate the load on the sender, especially if the peer list is lengthy.

4 Evaluation and Results

Currently, PalProtect has not been widely deployed, and we are unable to report on the behavior of a large scale deployment. However, our evaluation focuses on basic performance measurements of the system as deployed in our testbed. The evaluation of our current PalProtect system is mainly a feasibility study and focuses on the actual development of the plug-in so that is can be distributed to the WordPress com-

Table 1: Transmission test times for ten trials. Each trial represents the average of 1000 transmissions.

Trial	mean (s)
1	1.123607881
2	1.127622069
3	1.143602064
4	1.120167783
5	1.112963404
6	1.127018733
7	1.464225132
8	1.225920218
9	1.112342873
10	1.233204941

munity. There is much to do in terms of follow-up analysis of PalProtect, including analyzing privacy concerns and optimizing the routing of notifications and revocations.

4.1 Data Transmission

To see how feasible PalProtect is as add-on software to WordPress, we must first test the efficiency of the messaging process between two blogs. This measurement is crucial in seeing how quickly the plug-in can work before any more development is done. The average time it takes for messages to get from one place to another must be reasonably quick – otherwise the plug-in is only another component that will slow down the process of submitting a comment.

To quantify how long it would take for each transmission, we first took a median-length (see Section 4.2) spam comment and sent it 1000 times from one blog to another. We repeated this process for ten more trials. We then took the average time for each of the 1000 transmissions. The results are listed in Table 1

From these trials, we can see that the PalProtect plug-in transmits a spam message of typical length quite consistently. The typical time it takes for the transaction to complete is just about 1.1 seconds. This is a fairly reasonable base cost. Future work would involve analyzing the impact of an unreliable or broken network between the two peers. Since we are sending all the messages to one blog, the transmission time may be even faster since the target PalProtect system may be a bottleneck.

4.2 Space Efficiency

It is important to estimate how much space in storage the raw data and the signature will take up for every entry. First, we will need to find out how long a typical spam comment is. To do this, we had to manually browse the web looking for blogs that have fallen victim to spam comments. After browsing over twenty blogs, we had collected 161 spam comments. Many of the blogs we had found were very vulnerable, which led to many different types of bots and different types of spam. Though the number of blogs were not many in number, the different types of comments from each were sufficient.

The statistics of the lengths of these 161 spam comments were as follows:

1. **Mean**: 646.7081
2. **Median**: 154
3. **Standard Deviation1**: 1901.8820

From this data, we can conclude that the average length of the comment data will be around 650 characters, but half of the time it will be around 150 characters and below.

The next step is to analyze how much additional space each different signature type requires. For example, the MD5 hash signature will always be 128-bits simply because it is implemented that way. The Z-String will also be fixed at 256 characters. However, other methods, such as the URL list, can still vary a lot from comment to comment. It can range from having no URLs to having over 1000 characters of links to other websites.

4.3 Enforcement Time

The final performance concern is how quickly enforcement can occur; that is, how quickly the local signature database and matching algorithm operates.

The way that PalProtect currently implements enforcement, we expect that the performance grows linearly with time. This method is a simple traversal of every element in the signature list that compares a signature of the incoming comment with the current record. This operation is $O(n)$.

However, we can improve on the time complexity of the enforcement. Instead of traversing a list, PalProtect can maintain a hash table with entries for each signature in the database. This means that instead of having an $O(n)$ operation, enforcement can achieve $O(1)$ time complexity.

5 Conclusions

Collaborative security is an emerging area of research and a powerful tool against attackers whose activities are "globally loud but locally quiet." Such attackers seek to spread their activities over space and time so that they do not raise above a local threshold. Comment spam is one particular type of threat that can be addressed by a collaborative security system. We have presented PalProtect, a WordPress plug-in that identifies such spam and notifies its peers. At the cost of having a few community members or peers detect the spam, the entire group can be inoculated against future instances of that spam comment (or closely related variations thereof).

There remain a number of challenges for collaborative security systems. First, the utility of cross-domain information sharing is questionable when the domains sharing information do not share common interests. These sorts of incompatibility lead to larger questions of trust between peers. In addition, sufficiently large-size networks require elegant and scalable routing algorithms to help compress, store, and transmit information in a timely manner while minimizing the required bandwidth.

References

[1] D. J. Malan and M. D. Smith, "Host-Based Detection of Worms through Peer-to-Peer Cooperation," in *Proceedings of the* 3^{rd} *ACM Workshop on Rapid Malcode (WORM)*, November 2005.

[2] D. Moore, C. Shannon, G. Voelker, and S. Savage, "Internet Quarantine: Requirements for Containing Self-Propagating Code," in *Proceedings of the IEEE Infocom Conference*, April 2003.

[3] K. Anagnostakis, M. B. Greenwald, S. Ioannidis, A. D. Keromytis, and D. Li., "A Cooperative Immunization System for an Untrusting Internet," in *Proceedings of the* 11^{th} *IEEE International Conference on Networks (ICON)*, pp. 403–408, October 2003.

[4] H.-A. Kim and B. Karp, "Autograph: Toward Automated, Distributed Worm Signature Detection," in *Proceedings of the USENIX Security Conference*, 2004.

[5] S. Singh, C. Estan, G. Varghese, and S. Savage, "Automated Worm Fingerprinting," in *Proceedings of Symposium on Operating Systems Design and Implementation (OSDI)*, 2004.

[6] M. Costa, J. Crowcroft, M. Castro, and A. Rowstron, "Vigilante: End-to-End Containment of Internet Worms," in *Proceedings of the Symposium on Systems and Operating Systems Principles (SOSP 2005)*, 2005.

[7] V. Yegneswaran, P. Barford, and S. Jha, "Global Intrusion Detection in the DOMINO Overlay System," in *ISOC Symposium on Network and Distributed Systems Security*, February 2004.

[8] P. Lincoln, P. A. Porras, and V. Shmatikov, "Privacy-Preserving Sharing and Correlation of Security Alerts," in *Proceedings of the USENIX Security Symposium*, pp. 239–254, 2004.

[9] W. Du and M. J. Atallah, "Secure Multi-Party Computation Problems and their Applications," in *Proceedings of the New Security Paradigms Workshop*, pp. 11–20, September 2001.

[10] C. Krugel, T. Toth, and C. Kerer, “Decentralized Event Correlation for Intrusion Detection,” in *Proceedings of the International Conference on Information Security and Cryptology (ICISC)*, December 2001.

[11] M. E. Locasto, S. Sidiroglou, and A. D. Keromytis, “Application Communities: Using Monoculture for Dependability,” in *Proceedings of the* 1^{st} *Workshop on Hot Topics in System Dependability (HotDep-05)*, June 2005.