Search Worms

Niels Provos
Google, Inc.
niels@google.com

Joe McClain
Google, Inc.
jmcclain@google.com

Ke Wang
Google, Inc.
kewang@google.com

ABSTRACT

Worms are becoming more virulent at the same time as operating system improvements try to contain them. Recent research demonstrates several effective methods to detect and prevent randomly scanning worms from spreading [2, 13]. As a result, worm authors are looking for new ways to acquire vulnerable targets without relying on randomly scanning for them. It is often possible to find vulnerable web servers by sending carefully crafted queries to search engines. Search worms[1] automate this approach and spread by using popular search engines to find new attack vectors. These worms not only put significant load on search engines, they also evade detection mechanisms that assume random scanning. From the point of view of a search engine, signatures against search queries are only a temporary measure as many different search queries lead to the same results. In this paper, we present our experience with search worms and a framework that allows search engines to quickly detect new worms and take automatic countermeasures. We argue that signature-based filtering of search queries is ill-suited for protecting against search worms and show how we prevent worm propagation without relying on query signatures. We illustrate our approach with measurements and numeric simulations.

Categories and Subject Descriptors

D.4.6 [Operating Systems]: Security and Protection - Invasive software

General Terms

Security

[1]Worms that query external servers for targets have been labeled metaserver worms by Weaver et al. [14]. As we discuss worms that query search engines, we refer to them as search worms in this paper.

WORM'06, November 3, 2006, Alexandria, Virginia, USA.

Keywords

Internet worms, Search, Security, Signature, Index filtering

1. INTRODUCTION

As computer worms are becoming more common and spreading faster, operating systems like Microsoft's Windows XP are now shipping with mechanisms to slow down randomly scanning worms. Costa et al. show that it is possible to contain even fast-spreading worms like Slammer by instrumenting hosts to detect worms and broadcast self-certifying alerts to other end hosts [2]. These counter measures do not help if a worm can create a hit list of live targets without randomly scanning for them. An example of such metaserver [14] worms are worms that search networked databases for new targets. As web applications are becoming more common and sophisticated, the number of vulnerable web servers is increasing too. By carefully crafting search queries to search engines, an adversary can quickly create a list of vulnerable web servers. Automating this approach has led to the emergence of worms that spread by infecting targets found as a result of specific search queries. This hurts not only the owners of infected machines but also the search engine, as queries from search worms tend to be more expensive than queries from regular users.

In the following, we use the worm definition due to Kienzle et al. [4]: "A worm is malicious code (standalone or file-infecting) that propagates over a network, with or without human assistance." This definition allows us to include MyDoom.O in the list of search worms [12]. MyDoom.O is an example of a worm that requires human assistance to propagate. It sends itself to email addresses found by using a search engine. When a user executes the binary attached to the worm email, the worm infects the machine and searches for more email addresses. Santy, another search worm, does not require human intervention; it searches for web servers running a vulnerable version of phpBB and directly exploits them to run another worm instance [3].

In this paper, we discuss the unique properties of search worms and how they affect worm propagation. We analyze measurements taken from MyDoom.O and Santy outbreaks and discuss the impact of different parameters by employing numerical simulations. We present a framework based on Polygraph [8] which automatically detects the occurrence of new viruses and generates signatures that block them without incurring noticeable false positives. Although we show that signatures can stop a search worm from propagating, we argue that on top of stringent operational requirements: low false positives, high performance and fast turnaround,

the nature of search worms is ill-suited for signature-based protections. It is very easy for a search worm to completely change the search query and still obtain the same result set. Instead, we propose an algorithm that prevents worm propagation based on analyzing the result set: During indexing, we tag pages that belong to vulnerable servers or contain potential infection targets. If we detect search results that include a large fraction of tagged pages, we infer that the query is due to a search worm and return only results that have not been tagged. As a result, a search worm cannot spread because the search engine does not return any targets.

The remainder of the paper is organized as follows. In Section 2, we give an overview of related work. We discuss properties of search worms in Section 3 and provide data on two interesting incidents. Section 4 describes an architecture to mitigate search worm propagation. We discuss different factors effecting worm propagation in Section 5 and conclude in Section 6.

2. RELATED WORK

There has been a lot of research on containing worm propagation. Much of this research relies on heuristics that analyze network traffic for anomalous behavior [10, 13]. Mitigation can happen via rate limiting or by generating content-based signatures that completely stop a worm outbreak. Of several proposals for automatically generating worm signatures [2, 5, 8], Vigilante [2] is the most promising, as it does not require trusting participants and allows everybody to generate their own filters locally, based on vulnerability proofs.

The initial containment step of our worm mitigation architecture is adapted from Polygraph [8], a system for generating signatures for polymorphic worms. Polygraph works by extracting *tokens*, which are maximal length substrings occurring in a large fraction of suspected worm samples. These tokens are combined into signatures that can match the small, disjointed pieces of invariant structure of a polymorphic worm. Unlike Vigilante, it can operate by observing worm traffic without simulating or communicating with vulnerable servers.

This paper analyzes a different kind of worm that spreads using search engines to find new targets. Approaches for detecting them are different from more traditional worms as it is possible to stop the worm from spreading directly at the search engine without relying on a distributed architecture of worm containment systems.

3. SEARCH WORMS

Search worms are different from other metaserver worms because queries to search engines return only partial result sets. The result set is ranked by the relative importance of each site. To get more results, search worms change their queries by using different keywords, adding random numbers or walking deep into the result set. Nonetheless, due to the ranking inherent in the returned results, a search worm encounters many result collisions across subsequent queries which affect its propagation performance.

A search worm usually executes the following sequence of operations:

1. **Generate search query**: The purpose of the search query is to return as many distinct targets as possible. Search worms often come with a list of prepared queries. Each query may be responsible for a different set of results, for example, different version numbers of vulnerable software packages. The search query may contain other parameters that optimize the performance of the worm, such as increasing the size of the returned results or asking for results deep in the result set.

2. **Analyze search results**: Search engines often return superfluous information that needs to be pruned. A search worm looking for vulnerable hosts is going to ignore URLs that belong to the search engine itself or do not meet the expected URL format. The analysis step may also prune duplicate results such as host names or email addresses.

3. **Infect identified targets**: Based on the targets returned from the analysis step, the search worm attempts to exploit them. This usually involves reformatting a URL to include the exploit and bootstrapping code. The bootstrap phase allows the search worm to install multiple payloads, including itself, on the compromised target machine. The installation may happen in multiple steps and often relies on infrastructure already installed on the target. For example, variations of the Santy worm try to download themselves on the infected machine first via *wget*, *curl* and then *fetch*. Additional payloads often include applications that join the compromised machine in a bot network controlled by the adversary.

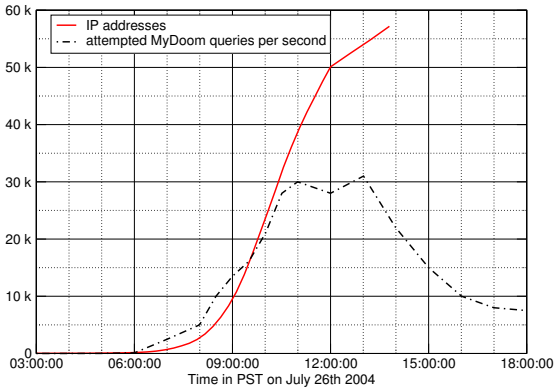

Figure 1: The graph shows the number of MyDoom.O search queries per second sent to Google on July 26th, 2004. It also shows the number of unique IP addresses sending MyDoom.O queries during that day, which can be used to approximate the number of infected hosts.

Next we turn our attention to our experiences with two different search worms. MyDoom.O requires human intervention to spread, but as our data shows that does not seem to be a hurdle. Santy, on the other hand, is the first search worm to spread completely automatically.

3.1 MyDoom

MyDoom is a worm that propagates via email. The email claims to be from a company's support department and contains an executable file as an attachment. When a user executes it, the worm gets activated and searches the local hard disk for email addresses of other users to infect. As a result, the worm propagates along the social network of the infected users. MyDoom.O improves on previous versions of MyDoom by increasing its branching factor. Instead of relying only on email addresses found locally, MyDoom.O uses the domain names of email addresses to search for more email addresses on Internet search engines. It first started spreading on July 26th, 2004 and managed to infect about 60,000 hosts in less than 8 hours. MyDoom.O used the following search engines, weighted by their respective probabilities: Google (45%), Lycos (22.5%), Yahoo (20%) and Altavista (12.5%).

Once MyDoom.O has received the search results, it analyzes them for email addresses and sends itself to all the addresses it finds. When a user opens a MyDoom.O email and executes the attachment, that machine, too, starts using search engines to find more email addresses.

Figure 1 shows the number of infected hosts and the number of MyDoom.O queries that Google received per second. The graph does not quite follow the standard SIR model as Google's anomaly detection system refused to serve most worm queries. During the peak of the outbreak, MyDoom.O infected machines attempted more than 30,000 queries per second.

3.2 Santy

The Santy worm surfaced on December 20th, 2004 and is the first search worm to propagate automatically, without any human intervention [3]. It is written in Perl and exploits a bug in the phpBB bulletin system that allows an adversary to run arbitrary code on the web server. To find vulnerable servers to infect, it uses Google to search for URLs that contain the string *viewtopic.php*. To infect a web server, Santy appends an exploit against phpBB2 to each URL extracted from the search results. The exploit instructs the web server to download the Santy worm from a central distribution site. Once the worm has been started, it asks the search engine for more vulnerable sites. In addition to the worm itself, all variants also download another payload connecting the infected machine to an IRC bot network.

Although, the number of infected machines stayed in the low thousands during the outbreak, the actual query traffic was larger as infected web servers were often well connected and ended up running multiple instances of the worm for each vulnerable virtual host. Santy initially targeted Google, but later variants also searched using Microsoft, Yahoo and Lycos. Santy has been more tenacious than MyDoom.O because most webmasters did not realize that their machines had been compromised and did not patch security holes in the web applications running on their servers. Even if a search engine prevented a particular instance of Santy from spreading by filtering search queries, it was easy for worm authors to find queries to circumvent the filtering and launch a new variant.

As bot networks are attractive and creating new Santy variants easy, we have seen a large number of modifications to the original worm. Using a honeypot to capture new outbreaks of Santy, we have graphed the dependencies between

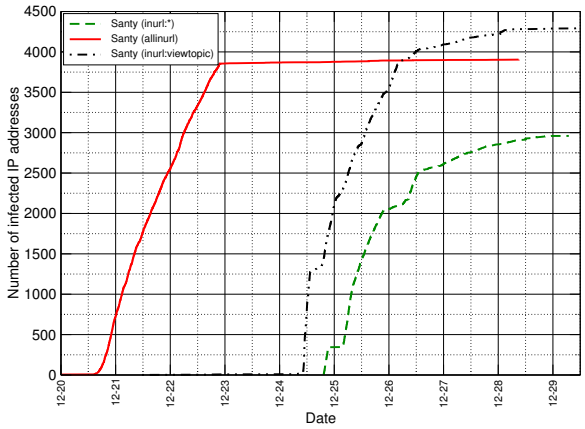

Figure 3: The graph shows a time-line of infected IP addresses for three different Santy variants in December 2004. Each variant manages to infect about four thousand different IP addresses. The propagation of the last two variants is overlapping in time.

different Santy variants in Figure 2. Each node in the graph represents a different Santy variant written in Perl and is labeled using its filename on the infected web server. To give an overview about how the Santy worm evolves along the time-line, we first connected each variant to the month and year in which it occurred, illustrated as the bar in the middle. We connected two nodes with an edge if their line difference computed via *diff* is minimal in respect to all other variants. As Santy is written in Perl, the number of changed lines is a reasonable measure. Across all Santy variants that we collected, the average number of line changes is about 484. The minimum number of changes lines is one and the maximum is 1689. Interestingly, the 10th, 50th and 90th quantile are at 56, 264, and 959 line changes respectively. The most common differences are changed search queries and distribution hosts. The graph shows that some variants of Santy have been continuously modified for over six months and that there are possibly many different adversaries launching new variants based on the disconnected components.

Figure 3 shows the number of IP addresses infected by the three earliest variants of Santy. The query signatures shown in the legend of the graph allow the the variants to be differentiated. Each variant managed to infect about four thousand different IP addresses. The graph shows a sudden plateau in the spread of the first variant. One possible reason might be that the worm managed to infected all susceptible hosts. However, the second variant reached slightly more IP addresses. A more likely explanation for this behavior is that an overload of the distribution site stopped worm propagation early; see Section 5 for simulation results.

4. WORM MITIGATION ARCHITECTURE

In the previous section, we discussed two different cases of search worms. From the search engine's point of view, it is difficult to differentiate between a search worm that requires human intervention to spread and one that spreads fully automatically. Either way, the search engine needs to quickly stop worm propagation to prevent machines from

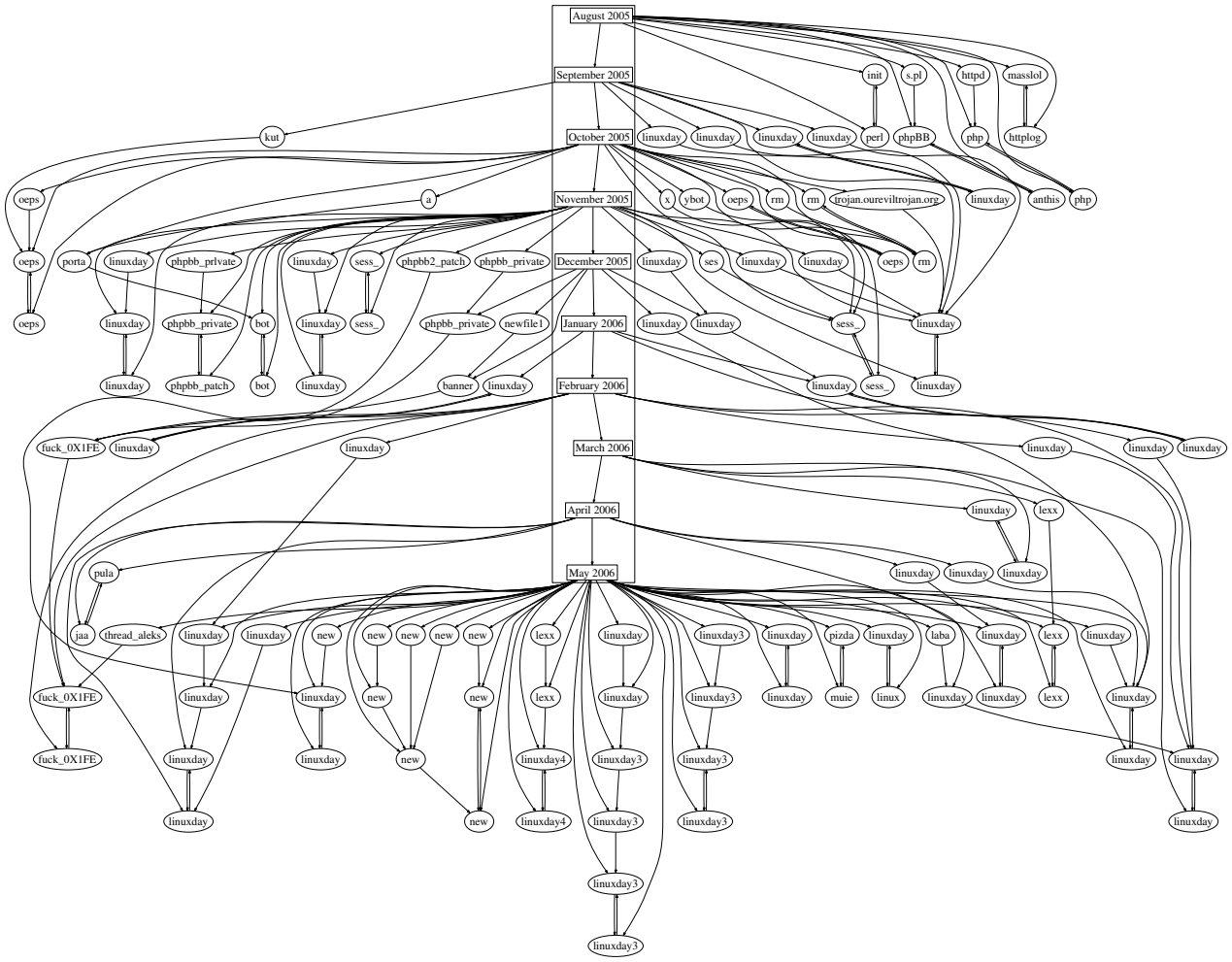

Figure 2: The graph shows the dependency between Santy variants from August 2005 to May 2006. Each node is labelled by the filename downloaded to the infected host. An edge between two nodes indicates that the differences in their code are minimal compared to all other variants. A time-line has been added to show how variants have evolved over time.

being infected and to avoid overload. The architecture of our worm mitigation system is divided into three parts:

- An anomaly detection step that automatically blocks parts of the worm traffic based on observing IP addresses.
- A signature generation step, based on Polygraph [8], that uses a sample of the blocked traffic and a sample of good query traffic to generate signatures that filter the current worm outbreak.
- Once the worm has been contained, the final step is to modify the search index so that subsequent variants against the same vulnerability will be unsuccessful.

An overview of the system is shown in Figure 4. Both the anomaly detection system as well as the signature generator provide data to the web servers for use in identifying and blocking bad queries.

4.1 Detecting Abnormal Traffic

To reduce the load on the search engine during the initial phase of a worm outbreak, we need to quickly classify the IP addresses responsible for abnormal traffic. Although the details of identifying an IP address as sending abnormal traffic are beyond the scope of this paper, here is one possible example: A search worm needs to carefully craft queries that are likely to return vulnerable candidates and also needs to be careful that subsequent queries return different results. This often implies that queries contain random numbers or version numbers. To counter this, we could create a map of frequent query words and use it to compute the compound probability for a query. A simple approach could compute the minimum probability for each word in the query: $\min(\mathcal{P}(q))$. An IP address that sends too many low probability queries gets flagged as abnormal. Further examples can be found in the existing literature [6, 9]. Once an IP address has been flagged as abnormal, we ask the user to complete a reverse Turing test [1] before answering the search query. If an IP address keeps sending queries without solving the reverse Turing test, we collect the corresponding queries to automatically create a candidate set for queries that should be filtered in general. Depending on the load of the search engine, we may decide to just sample the abnormal queries for the subsequent signature generation phase instead of challenging the user.

The anomaly detection step fulfills two important goals: it can be used to block the majority of worm traffic, buying additional time for the signature generation phase, and it also helps in collecting abnormal traffic samples for the signature generation step.

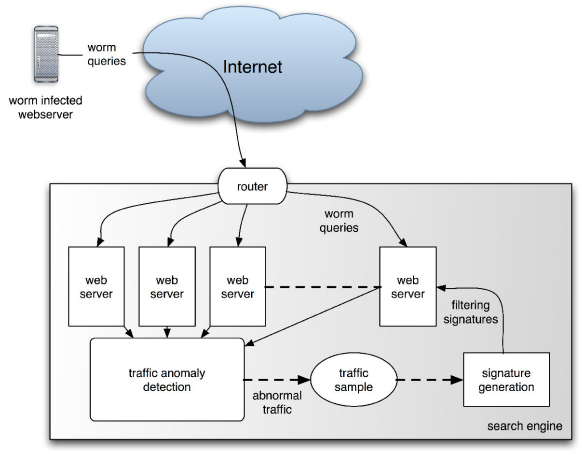

Figure 4: An aggregation server watches queries from IP addresses for abnormal behavior. Once abnormal traffic has been detected from a set of IP addresses, a sample of the requests is stored for signature generation.

4.2 Signature Generation

We create signatures in a fashion similar to Polygraph [8]. Polygraph extracts tokens from bad queries and uses these tokens to create signatures matching the bad traffic. It uses hierarchical clustering to merge signatures until a predefined false positive threshold is reached. False positives are computed by matching signatures against a good query set. Although Newsome et al. provide a good outline of the algorithms, they do not provide many insights into their performance.

In the following, we will describe the signature generation process for a Santy variant that emerged in 2005, outline changes to Polygraph, and discuss our implementation. We employed a good query set containing ten million sampled queries from the previous day. The bad query set consisted of about one million queries and was sampled continuously. We first took all queries in the abnormal query set and extracted tokens by finding the longest common substrings not covered already by a longer more frequent substring. We discarded all tokens not occurring in at least five percent of all queries. The main difference when applying Polygraph's algorithm to extract tokens to search queries is that search queries contain semantic boundaries along which we can split long query strings into smaller ones. This increased the speed of extracting the tokens and also improved the resulting signatures. We ran token extraction on a cluster of 85 2.4 GHz Intel Xeon machines which generated all tokens within a few seconds once the data files had been loaded into memory. For the query sets described above, the algorithm resulted in 56 different tokens. The top ten were: `[0-9]+`, `GET`, `/search`, `&num=`, `&start=`, `?q=`, `+[0-9]+`, `+-modules`,`+in`, and `+this`.

The hierarchical clustering step was more time consuming. Each unique tokenized query forms the basis of a signature[2]. We computed the cross product of all signatures by merging each signature with every other signature. Two signatures are merged by computing their longest common subsequence. For example, merging two signatures consisting of the tokens v, w, x, y, v and a, w, x, z, v respectively would yield the signature w, x, v[3]. The signature that results in the lowest false positive rate forms a new cluster into which the parent signatures are merged. The time complexity of the initial cross-product is $O(n^2)$ and the remainder of the clustering algorithm has a time complexity of $O(n \log n)$. In our experiment, we distributed the signature table across a cluster of 85 machines which completed hierarchical clustering after about 25 minutes. We could reduce the run-time by using smaller sample sets but our requirements for few false positives make that difficult. The following signature was generated as part of the final result set and matched about 75% of all bad queries without matching any query in the good sample set:

```
GET /search\?q=.*\+-modules&num=[0-9][0-9]+&start=
```

Although the signature above matches a high percentage of Santy queries, it is overly broad as can be seen when looking at the actual Santy queries shown in Figure 5. We can improve the signatures by terminating the clustering process earlier, which is governed by the threshold on the false positives. But, unfortunately, because the bad queries' structure is usually quite different from the normal ones', the generated signature will still have the same low false positive rate even when it is over-general. Scoring only based on the false positive rate, the merging step often leads to signatures that do not improve the detection rate noticeably but are more generic than necessary.

Newsome et al. motivated the Polygraph algorithm by arguing that even polymorphic worms have invariants due to protocol framing and the practical constraints of exploiting a vulnerability. This is completely different for search worms. Many different queries can lead to very similar result sets. For example, the following two queries: *"reply to topics in this forum" 1132*, and *"post subject" "powered by phpBB" 1132*, will give two result sets with high overlapping percentage by searching Google. Although, we employ signature-based schemes to prevent search worms from spreading, we have entered an arms race in which the adversary can quickly create new Santy variants that use different query terms; see Figure 2. This arms race peaked when we encountered IRC bots that took their Santy queries directly from the channel operator; see Appendix A.

4.3 Index-Based Filtering

To deal with the problem that multiple search queries may map to the highly similar set of result pages, we present a query independent algorithm that detects if search results are due to a search worm. The algorithm is based on the following observation: A search worm relies on a search engine to obtain a list of potentially vulnerable targets. If the search engine does not provide any vulnerable targets in the search results, the worm fails to spread.

As a search engine needs to crawl all pages shown in its index, we assume that it is possible to determine during the crawl phase, or while indexing, if a page is hosted on a vulnerable server or contains other potential targets such as email addresses. This decision can be based on the version of the web server, the web application or information available on the web page. Some of this information may

[2]A signature can be expressed as a regular expression but can also take other forms such as Bayesian signatures.

[3]Polygraph uses a modified Smith-Waterman [11] algorithm that we cannot describe in detail due to space constraints.

```
GET /search?q="View+previous+topic+::+View+next+topic"+8756+-modules&num=50&start=35
GET /search?q="vote+in+polls+in+this+forum"+7875+-modules&num=50&start=10
GET /search?q="reply+to+topics+in+this+forum"+5632+-modules&num=50&start=15
GET /search?q="Post+subject"+phpBB+6578+-modules&num=50&start=10
GET /search?q="delete+your+posts+in+this+forum"+9805+-modules&num=50&start=35
GET /search?q="post+new+topics+in+this+forum"+1906+-modules&num=100&start=30
```

Figure 5: The figure shows sample queries from a Santy outbreak in 2005.

also be provided externally; for example, a list of known vulnerable hosts. When creating the index, we tag all pages from vulnerable hosts or web applications with a piece of information that can be evaluated by the web server when it prepares the search results. Unfortunately, it is not possible to drop all pages from vulnerable servers completely from the index as they contain useful information users want to be able to access. To index billions of web pages efficiently, it is important to keep the size of the index small. In our case, we use a single bit to tag pages belonging to vulnerable servers. Another bit can be used to differentiate between pages containing infection targets rather than belonging to an exploitable server.

When a web server receives a search query, it retrieves all relevant results from its index servers. For each URL, the index result contains information about the vulnerability of the underlying host. The web server counts how many vulnerable URLs have been returned and to how many hosts they belong. Let n be the number of search results, e the number of vulnerable results and h the number of different web servers in the results. We can then choose thresholds α for the fraction of vulnerable results and β for the number of different hosts that we expect. If both $\frac{e}{n} \geq \alpha$ and $h \geq \beta$, then the current query belongs to a search worm. At this point, we can decide to not return any results at all. However, to reduce the collateral damage from false positives, we filter all pages from the search results that have been tagged as vulnerable before showing them to the user.

Empirical good values for α and β are around $0.3 - 0.7$ and $3 - 10$ respectively. The parameter β determines how many hosts a search worm can infect if it manages to stay below the threshold. The reason for using a threshold on the number of hosts at all is to reduce collateral damage when a user is looking for site specific information and restricts the search to a single site. Alternatively, we could use the host threshold β only if a query is navigational which requires reliable detection of navigational queries. Under the threshold-based approaches, we can adjust the thresholds accordingly to meet different effective level. Even when a well-designed, stealthy worm successfully evades the thresholds, the damage can be controlled within a limited small extension as the number of infected hosts will be kept low.

The benefit of an index-based approach is that it does not rely on classifying search queries as abnormal and that it can be implemented pro-actively. Whenever a new vulnerability for a web application has been discovered, we can classify it in the index before a search worm exploiting it has even been created. This approach has the limitation that we need to know the vulnerability before we can tag the index. For a zero-day attack, we rely on the mechanisms outlined in the previous sections to mitigate the impact of the search worm to buy some time. Once the vulnerability is understood, the index can be updated accordingly.

Because the index is pre-computed and the actual computations to detect a search worm are independent of the query, our approach is efficient and simple to implement. For search worms like MyDoom.O that are not directly infecting hosts, but are searching for vulnerable targets instead, we require an additional step that analyzes the content snippets returned with the search results. If we find that the content snippets contain infection targets like email addresses, we filter it out before returning the search result. This is necessary to avoid false positives as there are potentially many more web pages containing infection targets than there are vulnerable web applications.

5. DISCUSSION

The propagation rate of a search worm is influenced by many criteria. Santy serializes acquiring and infecting new targets. As a result, a Santy instance stops spreading if it gets blocked when either contacting the search engine or when infecting a host. Using Santy as an example, we discuss other factors that influence the propagation of search worms.

Instead of serializing search engine queries and infection attempts, a search worm may execute network requests in parallel. This avoids slowing down propagation when contacting slow or unavailable hosts. However, if a search worm spreads too quickly or sends too many search queries, it may overwhelm the search engine. At that point, no instance of the worm is going to be able to continue spreading. An intelligent search worm might want to throttle the number of search queries when it detects higher latencies on the search engine.

As search engines rank their results, there is a noticeable overlap in target hosts, i.e. even though a search worm may launch many different queries, each query might return only a small number of new hosts. Santy authors try to overcome this limitation by separating the search spaces. Common techniques include searching for distinct version numbers of a vulnerable web application or including random numbers in the query.

The overlap in search results slows down worm propagation as popular hosts become targets of numerous infection attempts. Even if the worm prevents multiple instances from running, the bootstrap code downloads the worm and other payloads for each attempted infection. This increases not only the load on the infected host but also the load on the distribution site. If the distribution site gets overloaded the worm ceases to spread. Some Santy variants have tried to cope with this behavior by giving each worm instance a fixed lifespan after which they stop spreading. Implications of this and other strategies have been analyzed by Ma et al. [7]

Interestingly, if a search engine applies anomaly detection techniques that do not block 100% of all worm queries, it may help the worm spread. Common anomaly detection techniques block IP addresses based on their behavior. This usually implies that a worm can make a few queries before the corresponding IP address gets blocked. Figure 6 shows a numerical simulation of the potential impact an anomaly de-

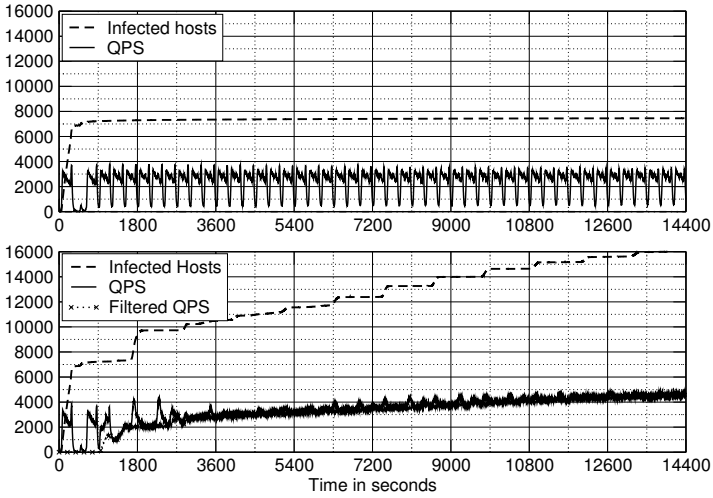

Figure 6: The upper graph shows the simulated propagation of Santy.C when a search engine does not employ an anomaly detection system. The lower graph shows the simulated propagation when an anomaly detection system prevents most worm queries from being successful. In the former case, the worm overloads the distribution host resulting in throttled propagation speeds. In the latter case, the anomaly detection system reduces the load on the distribution host by stopping most worm instances from spreading.

tection system can have on the propagation of Santy.C. We show the median results from thirty randomized executions. The upper graph, which simulates a search engine without employing an anomaly detection sytem, shows that worm propagation stops once the distribution site gets overloaded. The load on the distribution site is aggrevated by the fact that Santy.C. causes already infected hosts to download both the worm as well as the payload. The check for dual infection is contained in the payload itself rather than in the exploit to causes the download. The more machines are being infected, the more likely that already infected machines are returned as part of the search results. These machines keep the load on the distribution site high. At some point, it's not possible for new worm instances to start. On the other hand, the lower graph shows what happens when instances flagged by an anomaly detection system are prevented from searching. The infected hosts that are being prevented from searching, no longer cause additional downloads from the distribution host and the load on the distribution site is reduced. As a result, the worm is able to infect twice as many systems.

6. CONCLUSION

Worms no longer infect their targets by randomly scanning the Internet. They now also take advantage of the huge amount of information collected by search engines and spread by querying a search engine for new targets to infect. To demonstrate that this is a real problem, we have presented our experience with two different search worms: MyDoom.O and Santy. We have shown that signature generation in conjunction with anomaly detection can prevent a worm from overloading a search engine but that the approach is not effective in preventing a worm from spreading. We argue that a signature-based approach is ill-suited against search worms, as many different search queries can lead to the same result. That is, even after signatures have been applied to block a particular variant of a search worm, it is easy for adversaries to find queries that circumvent existing signatures.

To deal with this problem, we propose a solution that is query independent and classifies web pages as vulnerable if they belong to an exploitable server or contain potential infection targets. If a search query generates too many vulnerable results, we can detect that the search query belongs to a search worm and remove vulnerable results before showing them to the user. This approach is CPU efficient as it relies on pre-computation and is independent of the actual query.

7. ACKNOWLEDGMENTS

We thank Urs Hölzle and Vern Paxson for helpful suggestions and feedback. We also thank Monica Chew and the anonymous reviewers for their thoughtful comments.

8. REFERENCES

[1] A. L. Coates, H. S. Baird, and R. J. Fateman. Pessimal print: A Reverse Turing Test. In *Proceedings of the Sixth International Conference on Document Analysis and Recognition*, pages 1154–1158, September 2001.

[2] M. Costa, J. Crowcroft, M. Castro, A. Rowstron, L. Zhou, L. Zhang, and P. Barham. Vigilante: End-to-end containment of Internet worms. In *Proceedings of the 20th ACM Symposium on Operating System Principles (SOSP)*, October 2005.

[3] F-Secure. F-Secure Virus Descriptions: Santy. `http://www.f-secure.com/v-descs/santy_a.shtml`, December 2004. Accessed June 5th, 2006.

[4] D. M. Kienzle and M. C. Elder. Recent worms: a survey and trends. In *WORM '03: Proceedings of the 2003 ACM workshop on Rapid Malcode*, pages 1–10, New York, NY, USA, 2003. ACM Press.

[5] K.-A. Kim and B. Karp. Autograph: Toward Automated Distributed Worm Signature Detection. In *Proceedings of the USENIX Security Symposium*, August 2004.

[6] C. Kruegel and G. Vigna. Anomaly detection of web-based attacks. In *Proceedings of the 10th ACM conference on Computer and communications security*, pages 251–261. ACM Press, October 2003.

[7] J. Ma, G. M. Voelker, and S. Savage. Self-stopping worms. In *Proceedings of the 2005 ACM workshop on Rapid malcode*, pages 12–21, November 2005.

[8] J. Newsome, B. Karp, and D. Song. Polygraph: Automatically Generating Signatures for Polymorphic Worms. In *Proceedings of the 2005 IEEE Symposium on Security and Privacy*, pages 226–241, 2005.

[9] R. Sekar, A. Gupta, J. Frullo, T. Shanbhag, A. Tiwari, H. Yang, and S. Zhou. Specification-based Anomaly Detection: A New Approach for Detecting Network Intrusions. In *Proceedings of the 9th ACM conference on Computer and communications security*, pages 265–274, November 2002.

[10] S. Singh, C. Estan, G. Varghese, and S. Savage. The EarlyBird System For Real-time Detection of Unknown Worms. Technical Report CS2003-0745, UCSD, August 2004.

[11] T. Smith and M. Waterman. Identification of Common Molecular Subsequences. In *Journal of Molecular Biology*, volume 147, pages 195–197. 1981.

[12] Sophos. Sophos Virus Analysis: W32/MyDoom-O. `http://www.sophos.com/virusinfo/analyses/w32mydoomo.html`. Accessed June 4th, 2006.

[13] J. Twycross and M. M. Williamson. Implementing and Testing a Virus Throttle. In *Proceedings of the 12th USENIX Security Symposium*, Aug 2003.

[14] N. Weaver, V. Paxson, S. Staniford, and R. Cunningham. A Taxonomy of Computer Worms. In *WORM '03: Proceedings of the 2003 ACM workshop on Rapid Malcode*, pages 11–18, New York, NY, USA, 2003. ACM Press.

APPENDIX

A. SANTY FRAGMENT

Some variants of the Santy worm could change their search queries via IRC commands reducing the effectiveness of using signatures to filter search queries. The following code fragment illustrates this approach:

```
...
  if (\$funcarg =~ /^google\\s+(\\d+)\\s+(.*)\\s+(.*)/) {
  #sendraw(\$IRC_cur_socket, \"PRIVMSG \$printl :\\002[GOOGLE-SEARCH]
  #\\002 Scanning for unpatched phpBB for \".\$1.\" seconds.\");
  srand;
  my \$itime = time;
  my (\$cur_time);
  my (\$exploited);

  \$boturl=\$2;
  \$cur_time = time - \$itime;
  \$exploited = 0;
  while(\$1>\$cur_time){
     \$cur_time = time - \$itime;
     @urls=fetch2(\"\$3\");
...
```