Allergy Attack Against Automatic Signature Generation

Simon P. Chung and Aloysius K. Mok*

Department of Computer Sciences,
University of Texas at Austin, Austin TX 78712, USA
{phchung, mok}@cs.utexas.edu

Abstract. Research in systems that automatically generate signatures to filter out zero-day worm instances at perimeter defense has received a lot of attention recently. While a well known problem with these systems is that the signatures generated are usually not very useful against polymorphic worms, we shall in this paper investigate a different, and potentially more serious problem facing automatic signature generation systems: attacks that manipulate the signature generation system and turn it into an active agent for DoS attack against the protected system. We call this new attack the "allergy attack". This type of attack should be anticipated and has in fact been an issue in the context of "detraining" in machine learning. However, we have not seen a demonstration of its practical impact in real intrusion detection/prevention systems. In this paper, we shall demonstrate the practical impact of "allergy attacks".

Keywords: Automatic Signature Generation, Adaptive Response, Intrusion Prevention.

1 Introduction

With the proliferation of worms propagating at speed too fast for human intervention, automatic worm containment is increasingly looked upon as a solution. An important line of work in this area is the automatic signature generation (ASG) systems, which generate signatures to filter worm instances at perimeter defense. Similar to [3,8,10], our work focuses on certain weakness common to many ASG systems. In particular, we focused on a type of attack against many proposed ASG systems that we call the "allergy attack". While worm polymorphism renders ASG systems ineffective [3,8], the allergy attack allows attackers to easily turn the ASG systems (as well as the perimeter defense) into their agents for inflicting harm on the protected network by manipulating the ASG system so that traffic of their choice will be filtered at the perimeter defense of the target network. A vulnerability that turns ASG systems from an imperfect, but nonetheless harmless defense mechanism into an active threat to the protected network, is just as damaging to the usability of ASG systems as

* The research reported here is supported partially by a grant from the Office of Naval Research under contract number N00014-03-1-0705.

D. Zamboni and C. Kruegel (Eds.): RAID 2006, LNCS 4219, pp. 61–80, 2006.

compared to the very well addressed issues of worm polymorphism. The scope of the threat from allergy attacks is also much wider. For worm polymorphism, the problem is mostly limited to systems that use a single contiguous byte sequence from worm packets as signatures, and we are starting to see solutions to this problem. However, allergy attacks are found to be possible against ASG systems that employ other kinds of signature as well (e.g. [6]).

We note that the problem with allergy attack is similar to the "causative, indiscriminate availability" attack mentioned in [1]. However, the work in [1] focuses on the much higher level problem of attacking machine-learning based security mechanisms in a theoretical setting. While the authors of [1] have proposed many different means of abusing a machine-learning based system (e.g. inducing high false positives, evading detection), our study of allergy attacks is more specific on one particular issue, the viability of inducing high false positives in an ASG system in practice. To demonstrate the practicality of allergy attacks, we have experimented with one publicly available ASG system, and analyzed the algorithms of another 8 systems [1]. Our work, as presented in this paper complements the work in [1] by our experimental validation of allergy attacks. Another subtle difference between the work in [1] and ours is that while machine learning algorithms generate classification schemes based on a given set of features, ASG systems generally need to identify these features before applying any machine-learning technique. As our work and the work in [10] show, the feature extraction mechanism is an important avenue for attacking ASG systems that's not found in purely machine-learning based systems.

With the exception of the study in [1], our literature survey shows that the threat from allergy attack has received limited attention from the research community. As far as we can determine from the open literature, resilience against allergy attack is never a design objective for any published ASG systems. Yet we believe very strong guarantee on such resilience will be necessary if ASG systems are to be of any practical use. Our contributions in this paper are two-folded:

1. By defining allergy attack and demonstrating the attack against a typical ASG system (Autograph), we hope to draw attention to the threat posed by this type of attack.
2. By presenting our insight on what caused this vulnerability, we wish to help the design of future ASG systems to be resilient against allergy attacks. An understanding of what facilitates exploitations of the vulnerability will also help devising remedies for existing ASG systems.

In the next section, we will define what an allergy attack is. Then we will present some related work in Sect. 3. Our demonstration of allergy attack against Autograph will follow in Sect. 4. We will then describe a more sophisticated kind

[1] Courtesy of Professor Wenke Lee, we came upon the paper [1] that has just been presented on March 21-24, 2006. This paper anticipates theoretically our allergy attack but does not provide any empirical evidence of the viability of such attacks. We are also aware that the detraining of machine learning systems has been a serious concern to designers of military systems.

of allergy attack, the type II allergy attack in Sect. 5. We believe the type II attack allows us to defeat many simple defenses against allergy attacks. In Sect. 6, we will present our initial theory on the root of the vulnerability against allergy attacks, and factors that ease the exploitation of the vulnerability. Finally, we will conclude in Sect. 7.

2 Defining Allergy Attack

We start our discussions on allergy attack by defining it as follow:

> An allergy attack is a denial of service (DoS) attack achieved through inducing ASG systems into generating signatures that match normal traffic. Thus, when the signatures generated are applied to the perimeter defense, the target normal traffic will be blocked and result in the desired DoS.

In this work, we focus on the allergy attack against valid requests for services provided by the attacked network. Upon a successful attack, the signatures generated will block all instances of the target requests at the perimeter defense, and make the corresponding service unavailable to the outside world. As will be seen, it is also possible for the attacker to have very fine grain control over what services to be attacked (e.g., instead of blocking access to the entire web-site, the attacker may choose to make only particular pages unavailable).

Since all existing ASG systems work by observing traffic on the network, the only way for the attacker to manipulate vulnerable systems is by presenting them crafted packets. Packets crafted for an allergy attack should have the following properties:

1. The packets will be classified by the vulnerable system as suspicious, and will be used for signature generation.
2. The packets, when used for signature generation, will result in the desired signatures being generated.

We note that although it might appear that the problem can be easily solved by checking new signatures against some corpus of normal traffic, this turns out to be a non-trivial task to be done correctly and efficiently. First of all, the amount of memory needed for a corpus of normal traffic can be impractically high. The time needed to compare a signature against the corpus also makes this method infeasible. In fact, the experiments in [8] illustrates this point very well: with a corpus containing 5 days' worth of HTTP traffic, the signature generation process for a relatively small suspicious pool is reported as "under ten minutes". Furthermore, even though many ASG systems employ ways to eliminate false positives using normal traffic observed, most of these systems remain vulnerable to allergy attack (as we will show in Sect. 6). This is mainly because the mechanisms employed are designed to tackle "naturally occurring" false positives, not those intentionally produced by the attackers. Finally, as we will show in Sect. 5, there is a special kind of allergy attack that evades many corpus-based countermeasures, including the one mentioned above.

3 Related Work

In this section, we present some mainstream approaches for building ASG systems. Further details about particular ASG systems will be given in Sect. 4 and Sect. 6 when we illustrate how to attack those systems.

3.1 String-Matching ASG Systems

Among the proposed ASG systems, many employ simple byte sequence(s) as signatures [4,5,7,8,11,13,14]. Incoming traffic containing the byte sequences (or a large portion of it) will be considered malicious and dropped. The work of a string-based ASG system can generally be divided into two parts: first, worm packets are identified in the monitored traffic by some heuristic based approach, then invariant byte sequences are extracted from these suspicious packets as worm signatures.

3.2 Other ASG Systems

As many have noted [3,8], instances of polymorphic worms may not have invariant byte sequences long enough to be used as reliable signatures. In order to tackle this problem, some have proposed using other properties of suspicious packets as signatures. For example, the work in [12] used byte-frequency distribution for contiguous bytes in worm packets as signature. Krugel et al in [6] try to identify executable payload in worm packets, and extract common properties in their control-flow graph as worm signatures. Finally, Vigilante [2] extracts protocol frame values necessary for control hijacking to identify all worms exploiting the same vulnerability.

3.3 Allergy-Type Attack in the Literature

Even though the allergy attack has not been demonstrated in real systems in practice, we have found several brief mentioning of this type of attack in the following work [4,11,14]. The most detailed documentation of this potential problem can be found in [11]. We quote from Singh et al in [11]:

> Moreover, automated containment also provokes the issue of attackers purposely trying to trigger a worm defense - thereby causing denial-of-service on legitimate traffic also carrying the string.

In fact, the work in [11] is the only one that has explicitly mentioned the possibility of denial-of-service resulting from an allergy-type attack. In [4], the problem is referred to as "attackers deliberately submit innocuous traffic to the system", while Yegneswaran et al used the term "intentional data pollution" in [14]. More importantly, no practical solution has been proposed so far. The authors of [11] suggested "comparing signatures with existing traffic corpus - to understand the impact of filtering such traffic before we do so", which is infeasible due to the amount of time and memory required. Kim et al [4] proposed

"vetting candidate signatures for false positives among many distributed monitors", which can be defeated if all the participating sites are attacked in parallel (a similar approach is used in [13]. We shall present more details of these systems as well as the corresponding allergy attack in Sect. 6). Finally, Yegneswaran et al [14] resorts to human sanity check for the signatures, which basically defeats the purpose of speeding up the signature generation by avoiding human intervention. Furthermore, such sanity check may be infeasible when a large number of signatures are involved, and signatures resulting from an allergy attack are mixed with real worm signatures. Also, experience shows that real-world system administrators will simply turn off the ASG system if a manual check is required everytime a signature is generated.

4 Attacking Autograph: A Demonstration

In this section, we shall demonstrate the allergy attack against a real ASG system: Autograph. Both the design of our attack against Autograph and the results of our experiments will be presented. We use Autograph for our experiments because it is one of the very few ASG systems that we can work on. For other ASG systems, we either don't have access to them, or are incapable of collecting a normal traffic corpus necessary for their experimentation, due to privacy issues. Nonetheless, we find that Autograph has many properties typical in ASG systems vulnerable to allergy attacks (e.g. semantic-free signature generation, purely network-based "worm" detection, etc), and thus is sufficient for demonstrating the feasibility of allergy attacks. We understand that Autograph is a relatively old system, so we also outline the allergy attacks against some more recent ASG systems in Sect. 6. To give some background about how Autograph works, a brief description of Autograph is given in the Appendix.

4.1 Attacking Autograph

Our attack against Autograph is divided into two steps. In the first step, we induce Autograph into classifying the machines we control (our "drones") as scanners. In the second step, we simply use the drones to connect to machines in the protected network, and populate Autograph's suspicious pool for the target port with our attack packets. These packets are crafted such that the desired signatures will be generated when they are used for signature generation. To ease our discussion in this and the next section, we assume the target traffic to be an HTTP request for a protected web server. We stress once again that other types of traffic can also be targeted, and HTTP requests are chosen only because it appears to be the most direct way to inflict loss to the attacked organization.

Due to the simple heuristics used by Autograph, the first step can be easily achieved by requesting connections with many random IP addresses. For some networks, an easier and faster method is available; we can send out TCP connection requests with a combination of flags that never appears in normal traffic

(e.g. with both SYN and URG set). Since these requests are dropped or rejected by most networks, they will be considered failure by Autograph. Thus our drones will be classified as scanners with very few packets sent. This latter technique is actually employed in our experiments.

After being classified as a scanner, each drone will proceed with the second stage where crafted attack packets are sent to the target network over successful TCP connections. If no parts of the target request are blacklisted, we can simply put the entire target request in our attack packets. *Since the experiments in [4] show that Autograph has very low false positives, we believe it is very unlikely for any part of the target request to be blacklisted. In other word, our simple allergy attack will succeed most of the time.*

Nonetheless, let us consider the worst case scenario where all content blocks from the target request are blacklisted in the training phase. This will thwart the simple attack described above. However, this obstacle is circumventable. An important observation is that the target request is always partitioned in its entirety during the training phase, and this always results in the same set of content blocks. New content blocks that are not blacklisted may be generated if a fragmented target request is partitioned by COPP. For example, let the target request be the byte sequence $b_0b_1b_2...b_{i-2}b_{i-1}b_ib_{i+1}...b_n$, with $n - i > m$ and $i - 1 > m$. Suppose b_i is the last byte of the first content block generated by COPP. If the byte sequence $b_{i-2}b_{i-1}b_ib_{i+1}...b_n$ is presented to COPP, a content block starting with b_{i-2} will be generated. Since Autograph is not producing blocks of less than m bytes, the block will continue at b_i. More importantly, though this new content block contains bytes from two blacklisted blocks (the one starting from b_1 and b_{i+1} respectively), it is a substring to neither. As a result, the allergy attack against the target request will be successful if we use $b_{i-2}b_{i-1}b_i...b_n$ in our attack packets. Another point worth noting is that, the above strategy will remain effective even if requests similar to our target are also blacklisted. This is because these similar requests will result in mostly the same set of content blocks being blacklisted, due to the content-based nature of COPP.

Without knowing the configurations of the COPP (e.g. m, a, and B), we do not know where the boundaries of content blocks lie when the target request is partitioned in the training phase. In other word, we do not know exactly which fragmented target request will result in content blocks that overlap two adjacent blocks in the original partition. Nonetheless, we can approximate the above strategy by using random, fixed-length subsequences of the target request. With sufficient trails, some of these subsequences will result in new, non-blacklisted content blocks, and achieve our goal. Note that by using fixed-length subsequences with random starting points instead of random suffixes of the target request, we vary the last bytes of the various suspicious flows partitioned, and improve our chance of success. This is because COPP usually generates a content block with the last m bytes of the flow partitioned (unless the last content block ends exactly at the end of the flow).

Based on the above observations, the concrete design of the second step of our attack is as follow: we will divide our attack into different rounds, and in each round, all drones will pick the same NUM_SEQ random subsequences of length SEQ_LEN from the target request. This can be achieved by synchronizing all drones to start the round at roughly the same time (with synchronization error of up to a few minutes), and use the same seed for the same random number generator. Each drone will then send each chosen subsequence to the target network over NUM_REP connections destined at the target port. Each drone will start the next round of attack t minutes after the completion of the previous round. This is to make sure that the suspicious flows from the previous round have expired (i.e. removed from the suspicious pool). Obviously, a larger NUM_SEQ will reduce the number of rounds needed to achieve a successful attack. For our experiments, we use two drones, with NUM_SEQ=30, and NUM_REP=50. We emphasize that we use such a small number of drones only to ease our experiments, we don't see any technical difficulties in a 10-fold or even a 100-fold increase in the nubmer of drones. As for the value of SEQ_LEN, a proper choice of SEQ_LEN will significantly improve our chance of success. However, since the best value of SEQ_LEN depends on the unknown a and m of the attacked Autograph system, a trial-and-error process over multiple rounds is necessary. Nonetheless, our experiments show that for all reasonable values of a and m, it is very likely for the attack to succeed in just one round with SEQ_LEN being 80 to 160. Finally, the choice of t will depend on the *t_thresh* parameter of Autograph. With the default value of *t_thresh* being 30 mins, we can safely assume the actual value being less than 90 minutes, since a *t_thresh* value higher than 90 minutes can lead to a prohibitively large suspicious pool (a similar argument appears in [11]).

4.2 Experiments

To demonstrate the effectiveness of our attack, we have tested it against Autograph for the HTTP requests to the following three webpages:

1. http://www.cs.utexas.edu
2. http://www.cs.utexas.edu/users/mok
3. http://www.cs.utexas.edu/users/mok/cs372/Fall05/projects/lab1/index.html

We generate the target requests with internet explorer (IE). For each target request, we pick a set of 10 seeds for generating the random subsequences in 10 different rounds of attack. The same 10 seeds are then used for the experiments with SEQ_LEN at 40, 80, 120 and 160. This allows us to compare the effectiveness of our attack at different SEQ_LEN without being affected by the randomness in the seeds used. The effectiveness of our attack is measured by the average number of distinct signatures generated for each of the 10 rounds under the same SEQ_LEN[2]. Finally, to test how the different configurations of Autograph affect

[2] Even though any single signature generated will completely block out the target request, we believe the number of signatures generated will reflect the robustness of our attacks for different targets.

the effectiveness of our attacks, we repeat our experiments at different values of a (with a=16, 32, 64, and 128) and m (with m=16, 32, and 64), which is basically the range for a and m tested in [4]. For the other parameters of Autograph, we simply use the default values. We choose to focus our experiments on a and m because these two parameters have the most impact on the success of our experiments.

A point worth noting is that our attack is specific to the web browser used. In other word, the signatures generated will mostly filter out requests from IE only. Other web browsers (e.g. Mozilla) are thus unaffected. Nonetheless, a determined attacker can launch a separate attack for each popular web browser. Since the market is mainly dominated by a few web browsers, we believe this is not a major undertaking.

Instead of installing Autograph to monitor real traffic crossing an edge network's DMZ, we choose to perform our experiments offline by feeding Autograph with traffic traces captured separately at the two drones used. This will expose Autograph to exactly the attack traffic originating from the drones, as well as the reply from the attacked network that Autograph is supposed to be protecting, while ignoring all other traffic to/from the drones. As a result, we are presenting to Autograph only the "slice" of traffic that is relevant to our attack. This approach greatly simplifies our work, and allows us to test the same attack traffic under different configurations of Autograph.

A disadvantage of the above approach is that it prevents us from studying the effect of the background noise to our attack. For background noise, we are referring to the scanning activities that happen constantly on the internet, as well as small-scale worm outbreaks over the world. The effect of these events is mainly to populate the suspicious pool of Autograph with flows other than those from our attack. Nonetheless, we believe we can easily make content blocks from these flows an insignificant portion of the suspicious pool. By increasing NUM_REP to 200 and having 20 drones instead of 2, we can almost guarantee that content blocks from our attack packets will be sufficiently prevalent to be used as new signatures (each will have 4000 copies in the suspicious pool). Furthermore, even with the higher NUM_REP and increased number of drones, we believe our attack is still entirely feasible.

Finally, to validate the claim that our attack will remain effective even if the target requests and some related requests are blacklisted during the training phase, we populate the blacklist with all content blocks from the three target requests as well as the requests for the following 5 related pages:

1. http://www.cs.utexas.edu/users
2. http://www.cs.utexas.edu/users/mok/cs372
3. http://www.cs.utexas.edu/users/mok/cs372/Fall05
4. http://www.cs.utexas.edu/users/mok/cs372/Fall05/projects
5. http://www.cs.utexas.edu/users/mok/cs372/Fall05/projects/lab1

A separate blacklist is generated for each tested Autograph configuration by using the entire request to be blacklisted (instead of its subsequences) in the

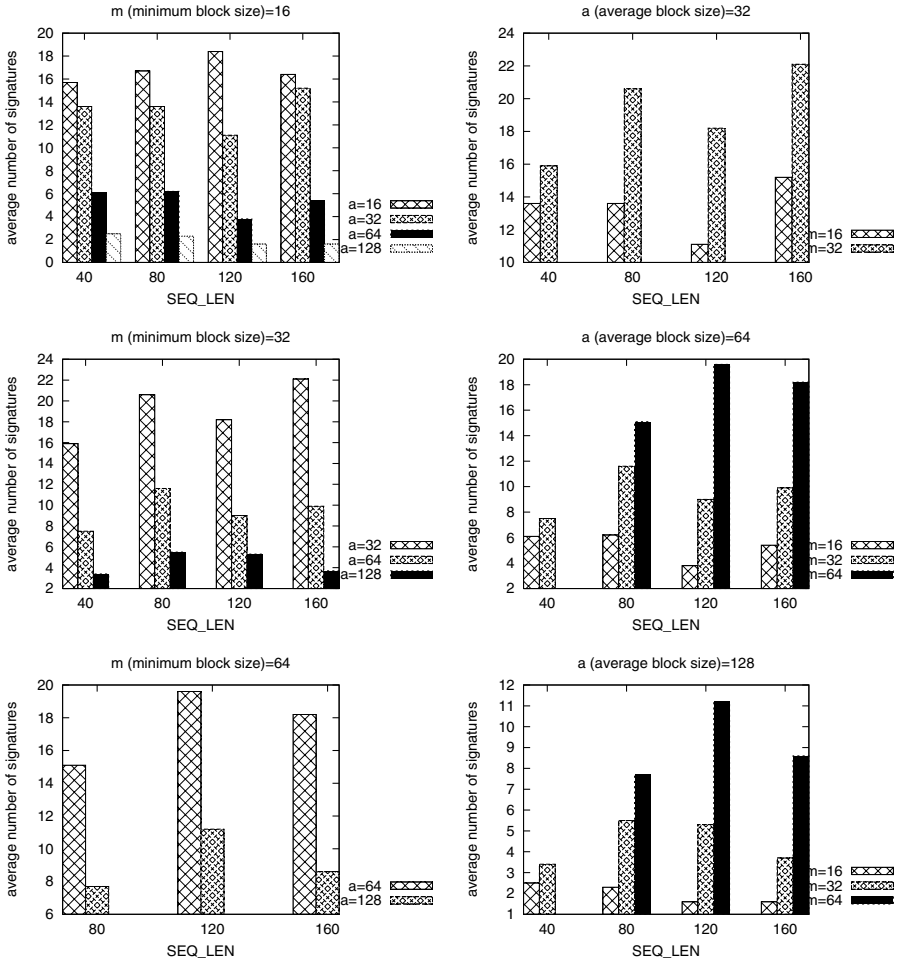

Fig. 1. The above figures show how the value of configuration parameters a and m of Autograph affect the effectiveness of our allergy attack at different SEQ_LEN. The effectiveness of our attacks is measured by the average number of distinct signatures generated in each of our 10 rounds of experiments. The column on the left shows the result of varying a while holding m constant, where the column on the right shows the effect of varying m while holding a constant. Note that no signatures will be generated when SEQ_LEN is smaller than m. Also note that we have only experimented on Autograph configurations with $a > m$.

attack described above. Every time Autograph generates a signature for our "attack traffic", we add it to the blacklist. We repeat the "attack" until no more signature is generated (i.e. all content blocks are blacklisted).

After describing the experimental setup, we will present our results on the first target request in Fig. 1. The results for the other two targets are very similar to those of the first one, and are therefore elided for brevity.

Our experiments show that the attack presented in Sect. 4.1 is very effective for all three target requests. At all combinations of SEQ_LEN, a and m where $SEQ\_LEN \geq m$, at least 8 out of the 10 rounds of our attack successfully induced Autograph into generating one or more signatures. Thus, we are confident that for any target request, at any reasonable configuration of Autograph, our attack will succeed in a small number of rounds, even if content blocks from the target requests (and some related requests) are all blacklisted.

Finally, observe that the effectiveness of our attack drops when a (the average block size) increases. This is because with a larger a, the target request will be represented by fewer (but longer) content blocks in the blacklist. As a result, it is less likely for any random subsequence from the request to cross the boundary of two adjacent blacklisted blocks and result in a successful signature generation. On the other hand, the effectiveness of our attack increases with m (the minimum block size), and this trend is more significant for larger SEQ_LENs. This may be due to the following behavior of COPP: a separate content block with the last m bytes of the flow will be created if the normal partitioning does not find a content block that ends at the last byte of the suspicious flow. We believe, the last block thus generated, as well as the first block for our random subsequence have the best chance of being a new, non-blacklisted content block that achieves our goal. This is because both of them don't start after a 2-byte subsequence that matches the breakmark B. Thus, a longer m will mean a longer last block, which in turn increases the chance for it to cross the boundary of two adjacent blacklisted blocks. Furthermore, longer SEQ_LEN will mean that the content of the first and the last block are significantly different, and improve the chance that they will produce two separate blocks that have not been blacklisted.

5 Type II Allergy Attack

Despite the success in our experiments against Autograph, the attack presented in Sect. 4.1 has two weaknesses. First of all, it is very likely that the signatures generated will match many other requests. In fact, we find that many of the signatures generated from our experiments will result in the filtering of all HTTP requests generated by IE. While this can be an advantage to some attackers, others may want the attack to be more specific. Furthermore, a direct allergy attack similar to that described in Sect. 4.1 will not be effective against ASG systems like Polygraph [8], which make use of a normal traffic corpus to avoid generated signatures from causing excessive false positives. *Even though the first problem only occurs when some part of our target request is blacklisted (which is a rare case), and the attempt to check generated signatures against a corpus of normal traffic appears impractical, let us once again assume the worst case scenario.*

To overcome the above weaknesses, we will propose a more sophisticated form of allergy attack: the attack against future traffic. We call this attack the type II allergy attack. This attack is first described in [11] as follow:

> However, even this approach may fall short against a sophisticated attacker with prior knowledge of an unreleased document. In this scenario an attacker might coerce Earlybird into blocking the documents release by simulating a worm containing substrings unique only to the unreleased document.

From the above description, we see that with a target that is different from the current traffic, the type II attack will be successful even if all generated signatures are matched against a normal traffic corpus. Due to the difference between the target traffic and the current traffic, some generated signatures will not match with anything in the corpus. Furthermore, if the characteristics that differentiate the current traffic and the target (i.e. the part that will be used in the generated signatures) is not common to all future traffic, the type II attack will also lead to a more specific DoS attack, which leaves other future traffic mostly unaffected.

As our survey showed, the level of sophistication required for the type II attack is actually not very high, since URLs on many websites evolve in an easily predictable manner. For example, the use of the date is common for news sites like cnn.com, while the use of ISBN numbers are common among queries to amazon.com. The URLs for Microsoft security bulletin which provide information about vulnerabilities related to Microsoft products contain the id number of the vulnerability (e.g. MS05-053). All these elements can be used for the type II allergy attack. Furthermore, the common use of web caching information in HTTP requests and responses makes many general HTTP traffic possible targets of the type II attack (for example, it seems possible for "poisonous" signatures that filter requests with the "If-Modified-Since" field being any value in the next 30 days to block all pages updated in the coming month).

Finally, we emphasize that any mechanism to avoid false positive by checking signatures against normal traffic only when they are generated (as in EarlyBird [11] and Polygraph [8]) will be defeated by the type II attacks, since any signatures from such attack will have zero false positive at the time of generation.

6 What Makes an ASG System Vulnerable and Exploitable

In this section, we will present some insight on the allergy attack, based on our study of different ASG systems. In particular, we will try to answer the following questions:

1. What are the properties that make an ASG system vulnerable to allergy attacks?
2. What are the properties that make a vulnerable ASG system easily exploitable?

To better illustrate our ideas, we will give examples of ASG systems with the properties concerned, and show how those properties guide the design of allergy attacks against vulnerable systems.

6.1 Semantic-Free Signature Generation

We believe the root of the vulnerability against allergy attacks lies at the semantic-free signature generation process. By semantic-free, we mean signatures are generated without considering how the properties matched by the signatures contribute to successful worm activities. In other word, the different components of a worm (e.g protocol frame for control hijacking, filler bytes for buffer overflow, return address used to direct control to worm payload, or the worm payload itself) are treated the same in a semantic-free signature generation process. This property makes it possible for the vulnerable ASG system to confuse part of the targetted traffic as an invariant property of a "worm", and use it as a "worm signature", and thus is a precondition for successful allergy attacks. Now let's consider some example ASG systems with this property.

Honeycomb: As one of the earliest ASG systems, Honeycomb [5] uses honeypots to collect worm packets, and generates signatures by finding the longest common substring (LCS) among the collected packets. As we can see, the signature generation process makes no attempt to find out how that longest common substring contribute to a successful attack. Attack against Honeycomb is similar to the one presented in Sect. 4.1, but is simpler. The attacker only needs to establish connections with random IP addresses, and send the request they want to have filtered over the connection. By repeating this a large number of times, the attacker will eventually establish multiple connections with the honeypot, and populate the suspicious pool with enough of his attack packets. The LCS-based signature generation will then generate signatures based on the target request in the attack packets.

EarlyBird: EarlyBird [11] is an ASG system designed to be efficient in both time and space, so that it can process traffic on high-speed link in real time. EarlyBird extracts prevalent 40-byte sequences that both originate from, and are destined to significantly diverse IP addresses as signatures. Once again, the signature generation ignores how the signature byte sequences affect the target host. Thus an attack strategy similar to the above applies to EarlyBird as well. Attack packets containing the target request alone will be sent to a large number of addresses in the target network (30 for the default configuration in [11]). In order for the attack to be successful, those packets should also appear to come from a diverse set of source addresses (30 again). Since EarlyBird performs very limited flow reassembly and no real connection is needed for a source address to be counted, we believe the source addresses can be spoofed. Even if more accurate flow reassemble process is employed and real connection is needed, the number of machines that the attacker has to control still appears to be insignificant for hackers nowadays.

In addition to understanding the role of different parts of the worm packet in the worm's activity, and make use of this information in signature generation, it is also very important to confirm that the identified worm components are "functional". If the signature is generated using the payload of the worm, it is

important to make sure that the "payload" is at least executable. If the signature generation utilize the return address used in a control hijacking, then it is necessary to check whether that address will result in the execution of worm payload. Without these sanity checks, the more sophisticated signature generation process will only complicate the design of allergy attacks, but will not stop them. Now let's consider the system presented in [6] and TaintCheck [9] as examples.

CFG-based Signature Generation: The system proposed in [6] is very similar to EarlyBird. However, instead of using 40-byte sequences as signatures, the system employs prevalent executable-code fragment that appears in packets with diverse source and destination. In particular, a code fragment is considered prevalent if its structurally equivalent variants are found in many packets (i.e. a byte-by-byte match is not necessary). With this new type of signatures, any packet that contains executable code with the same structure as the signature code fragment will be dropped. Signatures thus generated will then be resilient to certain polymorphic techniques (like register renaming and instruction substitution). However, the system in [6] does not verify that the executable code used in the signature is indeed a worm payload. In fact, any executable code in a suspicious packet can be used as a signature, even if it does not correspond to any worm activity, or will never be executed by the attacked host at all. As a result, it is possible to launch an allergy attack against the system in [6] to block all packets containing executable code of the attacker's choice. The attack is basically the same as that against EarlyBird, the only difference is that code fragments from the target executable are used in the attack packets. Even though this attack is ineffective against services offered by the protected network, it may be used to prevent hosts from downloading patches for vulnerabilities (which are usually packaged as ".EXE" files), or worm removal tools.

TaintCheck: TaintCheck [9] is a novel intrusion detection system that uses dynamic taint analysis to keep track of tainted data, i.e. data that originates or is derived arithmetically from an untrusted input. An alert is generated whenever the tainted data are used in an unsafe way, e.g. used as a jump address. Furthermore, TaintCheck can obtain the value of tainted data that is used for unsafe operations. In an injected code attack, this will mean the value used to overwrite a function pointer or return address. In [9], Newsome et al suggest using the most significant three bytes of this value as a signature for attacks exploiting the same vulnerability. The evaluation in [9] shows that this preliminary signature generation scheme is very effective in detecting attacks and results in low false positives. However, if the overwritten value is used to filter incoming traffic without checking whether it really leads to the eventual execution of the worm payload, the system will be vulnerable to allergy attacks. In order to block out the target request, the attacker simply modifies a real control-hijacking attack to overwrite the corresponding function pointer or return address with a 3-byte sequence in the target request.

6.2 Imperfect Suspicious Packet Detection

Just as a buffer overflow vulnerability does not always allow a successful attack, not every vulnerability against allergy attacks is exploitable. In this and the next section, we will present two factors that help the exploitation of a vulnerable ASG system.

As we see in the previous discussion, actual signature generation usually follows a detection process in which suspicious traffic are identified as the "raw material" for signatures. If this detection process may misclassify innocuous packets even in the absence of allergy attacks, it would appear to be easier for the attacker to populate the suspicious pool with his crafted packets. From our experience, ASG systems that rely on purely network-based detection are more vulnerable to false positive than those that employ some form of host-based detection. For example, Autograph, Honeycomb, EarlyBird, and the CFG-based system in [6] all employ detection mechanisms with non-zero false positive. As seen in our previous discussion, attacking these systems is relatively easy, all the attacker has to do in order to get his packets into the suspicious pool is to send out packets to/from the right addresses. On the other hand, attacks are much more complicated against systems like TaintCheck which have zero-false positive detection mechanisms. Actual attacks against the target systems are necessary to "feed" the signature generation process with the crafted packets. This is obviously more complicated and risky for the attacker. Now let's consider another example that employs a supposedly zero false-positive detection mechanism: FLIPS.

FLIPS: FLIPS [7] is a system that generates signatures to filter HTTP requests. It uses a network-based anomaly detection system called PAYL to identify suspicious packets. PAYL detects anomalous packets by using a normal profile that describes the byte-frequency distributions for normal traffic of different length and destination port. Any packet which shows significant deviation from the profile will be labeled suspicious. In FLIPS, all suspicious requests are cached. FLIPS also employs a host-based intrusion detection system called instruction set randomization (ISR). In addition to detecting attacks with zero false positive, the ISR also identifies the beginning of the attack payload. When ISR detects an attack, FLIPS will copy the first 1KB of the attack payload. The memory copied will be matched against the cached suspicious request based on a similarity score computed as $2C/(S1+S2)$, where C is the longest common substring (LCS) between the packet and the captured payload, S1 and S2 are the length of the two string being compared. The request most similar to the payload will be used as the signature of the worm. Any incoming request that is sufficiently similar to the signature will then be dropped.

Now let us consider an attack with the first target request as in Sect. 4.2 (which is 475-byte long). The attack involves sending two attack packets at the same time. The first packet is the one to be used as the worm signature at the end. This packet is constructed by appending to the target request a byte sequence that we call "gibberish". The gibberish is intended to

make the packet suspicious to PAYL. In our attack, we will have a 100-byte gibberish, all filled with a byte that rarely occurs in normal HTTP requests. This should make the packet sufficiently anomalous to be cached by FLIPS. The second packet contains a real code injection attack (e.g. the one from Code-Red), with the content of the first packet appearing at where payload should be placed (this makes the second packet around 1050 byte long). Once this second packet triggers the alarm from ISR, the cache will be searched for the suspicious request responsible. With similarity computed as described above, the request from the shorter first packet will achieve a higher similarity score of 0.73 (when compared to the 1KB "worm payload" identified by the ISR, which contains most of the 575 bytes in the first attack packet, and whatever follows in the memory when the attack is detected). Thus this first request will be used as the signature, and filter all instances of the target requests in future traffic (the similarity score between the target request and the signature is 0.90, which is much higher than the threshold used in [7]).

As a general observation, zero false-positive detection mechanisms cannot make an ASG system immune against allergy attacks, but it does force the attacker to employ modified control hijacking techniques, which makes the attack far more complicated. We also note that the attacker cannot directly take over the target system with the control hijacking code used in the allergy attacks; this is because the hijacking will be detected and stopped.

6.3 Independent Detection and Matching

Another property of ASG systems that eases the design of allergy attacks is the use of independent properties in the detection phase and the signature generation phase. Consequently, the properties of the suspicious packets that are used to filter future traffic are totally different from the properties that make those packets suspicious at the first place. Therefore traffic filtered by those signatures may appear completely innocuous to the detection mechanism.

The above property avoids any conflict between the construction of packets that will produce the desired signatures being generated when used for signature generation and getting those packets into the suspicious pool. This gives the attackers a lot of freedom. Example systems with such properties include Autograph, Polygraph, the CFG-based ASG system in [6], and systems that employ honeypot for collecting worm traffic in general. Another example of ASG system with this property is from [13].

PAYL-based ASG systems: In [13], two signature generation schemes are proposed. The first scheme, called ingress/egress correlation uses PAYL to identify suspicious ingress and egress traffic (with a separate normal profile for each direction). As in FLIPS, suspicious ingress packets are cached. Upon detection of a suspicious egress packet, the ingress cache is searched for a sufficiently similar packet destined to the same port. Similarity is measured either by string equality or longest common substring/subsequence (LCS/LCSeq). If a matching packet

is found, the part of the packet that gives the match (the entire packet if string equality is used, the longest common substring/subsequence if LCS/LCSeq is used) will be used as the signature.

In the second scheme, anomalous payload collaboration, different sites collaborate to identify new worms. The participating sites will compare suspicious ingress packets identified by their local PAYL with each other. If the suspicious packets are found to be similar, those packets are used as signatures to detect new worms. However, Wang et al are not very clear in [13] about how the new signatures are matched against traffic in either schemes. We will assume LCS/LCSeq as above to be used for similarity measure.

Note that while PAYL classifies packets based on byte-frequency distribution, signatures are generated, and matched against normal traffic by a totally different mechanism. As in our attack on FLIPS, we can make PAYL classify a packet anomalous with a short sequence of gibberish, with the majority of the packet made up of our target request (which will be matched against other traffic).

To attack the ingress/egress correlation mechanism, the attacker needs to control one protected machine. If only web servers are protected (which seems to be suggested in [13]), the attacker will need to compromise a web server in the protected network. The attack packet used here has the same structure as that against FLIPS: target request followed by gibberish. This attack packet is to be sent both to and from the comprised web serve, both destined to port 80. Both packets will be marked suspicious by PAYL (due to the gibberish), and the consequent correlation will make the entire attack packet a new signature (if we use a different byte for the gibberish in the egress and ingress packets, only the target request will be used in the signature). As before, the gibberish is only a small portion of the entire signature, and thus all future instances of the target request will be considered very similar to the signature and filtered.

The attack against anomalous payload collaboration is similar but much simpler. The same attack packet will be used. However, this time it will be sent to different networks in the collaborative scheme. Once again, PAYL will mark the packets as malicious, and since the same packet is seen at all collaborating sites, it will be used as a worm signature. The new signature will then achieve the expected DoS.

6.4 Vigilante: A Non-vulnerable ASG System

Vigilante [2] employs two zero false-positive mechanisms to detect attacks, namely the non-executable pages and dynamic dataflow analysis. The former technique allows injected code attack to be detected, while the latter is very similar to the dynamic taint analysis used in TaintCheck. Upon detecting an attack, the malicious input that results in the detection will be identified. The attack will then be replayed in a sandbox environment with the control flow and data flow of the attacked process recorded until the point where the attack is detected. A filter is then generated by computing the precondition of the input that leads to the recorded control and data flow in the attacked process.

We now consider whether Vigilante has any of the problematic properties listed above. First of all, we note that the signature generation process in Vigilante is semantic-based. By computing the precondition of the input that results in the control and data flow observed in a positive detection, Vigilante is effectively identifying the protocol frame necessary for a successful attack. Furthermore, both detection mechanisms have zero false-positive rate. Finally, for both detection mechanisms, the detection and the signature generation process are based mostly on the same property of the input, namely, properties that bring the protected system to the state where the attack is detected.

From the above analysis, it appears that Vigilante should not be vulnerable to allergy attacks. The filters generated by Vigilante indeed identify inputs that result in dangerous state changes in the destined system and nothing else. In general, systems that employ some host based detection mechanism are more difficult targets for allergy attacks. First of all, host based detection usually has a lower false positive rate than a purely network-based mechanism. Secondly, a host based mechanism can provide better information about how different parts of the malicious input correspond to the different components of the worm.

7 Conclusions

In this paper, we have presented the allergy attack, an attack against automatic signature generation (ASG) systems that has been anticipated theoretically but not demonstrated in practice. We start by defining allergy attacks as DoS attacks which result in normal traffic to the protected network being dropped by perimeter defense. This is achieved by inducing the ASG system in generating signatures that match the target traffic. When these signatures are deployed to the perimeter defense, the expected DoS will occur. In our discussion, we focused on the DoS against web service, which appears a most direct way to cause damages with allergy attacks. We then demonstrate the allergy attack against a well known ASG system: Autograph. We also analyze how similar attacks can be successfully mounted against other implemented ASG systems. We believe the vulnerability roots from the use of semantic-free signature generation process, i.e., the generation of signatures without considering how the properties matched by the signatures map to successful worm behavior. Two factors that facilitate the exploitation of this vulnerability are the imperfections in the mechanism used to identify suspicious packets, and the use of independent properties in the detection phase and the signature generation phase. In our future work, we plan to test some of the proposed attacks against actual implementations of other analyzed ASG systems, and to study the effectiveness of defending against type II allergy attacks with a normal traffic corpus[3].

As compared to the well studied issues with polymorphic worms, we believe the allergy attacks present a more pressing problem to practical ASG systems.

[3] Parties capable of experimenting with any studied ASG systems are welcome to collaborate with us, or verify our outlined attacks independently.

While polymorphic attacks will render many existing ASG systems totally useless, allergy attacks can turn them into real harm to the protected network. The cost of designing and launching an allergy attack is also much smaller than that of a effective polymorphic worm. The effect of an allergy attack can also be more long lasting than other common DoS techniques, since the target traffic will remain blocked until all the "poisonous" signatures are removed.

Finally, we will note that the scope of allergy attacks (or attacks with similar flavor) is not limited to ASG systems. As our defense evolves to react to attacks by modifying the state of the protected systems, the attackers may deliberately trigger the defense to modify the system's state in a way favorable to them. Defense designed under the old assumption that the attackers always try to evade detection may then be manipulated to serve the attackers' purpose.

References

1. M. Barreno, B. Nelson, R. Sears, A. D. Joseph, and J. D. Tygar. Can machine learning be secure? In *Proceedings of the ACM Symposium on InformAtion, Computer and Communications Security (ASIACCS2006)*, Taipei, Mar 2006.
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 20th ACM Symposium on Operating Systems Principles*, Brighton, Oct 2005.
3. J. R. Crandall, S. F. Wu, and F. T. Chong. Experiences using minos as a tool for capturing and analyzing novel worms for unknown vulnerabilities. In *Proceedings of GI/IEEE SIG SIDAR Conference on Detection of Intrusions and Malware and Vulnerability Assessment (DIMVA) 2005*, Vienna, July 2005.
4. H. Kim and B. Karp. Autograph: Toward automated, distributed worm signature detection. In *Proceedings of 13th USENIX Security Symposium*, California, August 2004.
5. C. Kreibich and J. Crowcroft. Honeycomb - Creating Intrusion Detection Signatures Using Honeypots. In *Proceedings of the Second Workshop on Hot Topics in Networks (Hotnets II)*, Boston, November 2003.
6. C. Krugel, E. Kirda, D. Mutz, W. Robertson, and G. Vigna. Polymorphic worm detection using structural information of executables. In *Proceedings of Eighth International Symposium on Recent Advances in Intrusion Detection (RAID2005)*, Seattle, Sept 2005.
7. M. E. Locasto, K. Wang, A. D. Keromytis, and S. J. Stolfo. Flips: Hybrid adaptive intrusion prevention. In *Proceedings of Eighth International Symposium on Recent Advances in Intrusion Detection (RAID2005)*, Seattle, Sept 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*, Oakland, May 2005.
9. J. Newsome and D. Song. Dynamic taint analysis for automatic detection, analysis, and signature generation of exploits on commodity software. In *Proceedings of 12th Annual Network and Distributed System Security Symposium (NDSS 05)*, Feb 2005.
10. R. Perdisci, D. Dagon, W. Lee, P. Fogla, and M. Sharif. Misleading worm signature generators using deliberate noise injection. In *Proceedings of The 2006 IEEE Symposium on Security and Privacy*, Oakland, May 2006.

11. S. Singh, C. Estan, G. Varghese, and S. Savage. Automated worm fingerprinting. In *Proceedings of 5th Symposium on Operating Systems Design and Implementation*, California, December 2004.
12. Y. Tang and S. Chen. Defending against internet worms: a signature-based approach. In *Proceedings of 24th Annual Joint Conference of the IEEE Computer and Communications Societies*, Florida, July 2005.
13. K. Wang, G. Cretu, and S. J. Stolfo. Anomalous payload-based worm detection and signature generation. In *Proceedings of Eighth International Symposium on Recent Advances in Intrusion Detection (RAID2005)*, Seattle, Sept 2005.
14. V. Yegneswaran, J. T. Giffin, P. Barford, and S. Jha. An architecture for generating semantics-aware signatures. In *Proceedings of 14th USENIX Security Symposium*, Maryland, Aug 2005.

A Appendix

Autograph is a string-matching system that generates worm signatures by monitoring traffic crossing an edge network's DMZ. Since the Autograph prototype available only handles TCP packets, we assume all traffic to be under the TCP protocol. Signatures generated by Autograph are destination port specific, i.e. only traffic destined to the corresponding port will be matched against a signature.

Autograph processes traffic in two stages. In the first stage, Autograph identifies scanners by recording IP addresses that made more than *s_thresh* unsuccessful connection attempts to the protected network. A connection attempt is considered unsuccessful if it times out without any reply received, or it got reset before completing the TCP handshake. In addition to the IP address, Autograph will also record the destination port targetted by all the failed connections from a scanner. Afterwards, all the TCP packets from successful connections originating from a scanner address and destined to the recorded port will undergo flow reassembly. The resulting suspicious flows will be recorded in a suspicious pool. With enough flows in the pool that are destined to the same port, Autograph will start the next stage of processing: signature generation.

In the signature generation stage, Autograph will divide the suspicious flows into content blocks, and find the set of most prevalent blocks. The process is greedy, and the block with highest prevalency will be picked first. Autograph will keep adding blocks to the set until a pre-configured portion of suspicious flows contain one or more blocks from the set. Signatures will then be generated for each block in the set, with the entire block being the byte sequence that will be matched against future traffic destined to the port for which signature generation is invoked.

For dividing flows into content blocks, Autograph employs the COnetent-based Payload Partitioning (COPP) technique. The COPP partitions suspicious flows into non-overlapping, variable-length blocks by computing the Rabin fingerprint of every 2-byte subsequence in the flow, starting from its beginning. The 2-byte subsequence marks the end of a content block if it matches B, i.e. its fingerprint r satisfied the equation $r = B \ (mod \ a)$, where B is a predetermined

breakmark, a is a configurable parameter that controls the average block size. Due to the content based nature of COPP, a similar set of blocks will be generated even if bytes are added to or deleted from the worm payloads. This helps Autograph to generate signatures that filter different instances of a polymorphic worm.

To avoid overly specific or overly general signatures, Autograph bounds the size of content blocks generated between m bytes and M bytes (with m and M configurable). In other word, Autograph will not end a content block at a 2-byte subsequence that matches B if that results in a block shorter than m. Instead, Autograph will search for the next matching 2-byte subsequence. Similarly, any content block that reaches M bytes long will be terminated. Autograph also avoids using content blocks in flows that originates from fewer than a configurable *source_count* number of sources for signatures. This prevents generating signatures for normal traffic from misconfigured, but benign hosts. Finally, Autograph employs a blacklisting mechanism which prevents subsequences of any blacklisted byte sequences from being used as signatures. In [4], the blacklist is generated in a training period where all signatures generated are manually checked for false positives. Signatures deemed to match normal traffic will be added to the blacklist. This prevents generating signatures for normal traffic that Autograph normally misclassifies.