LECTURES NOTES IN ELECTRICAL ENGINEERING 30

Vasilios Siris
Sotiris Ioannidis
Kostas Anagnostakis
Panagiotis Trimintzios
Editors

Proceedings of the 3rd European Conference on Computer Network Defense

Springer

Proceedings of the 3rd European Conference on Computer Network Defense

(continues after index)

Vasilios Siris • Sotiris Ioannidis •
Kostas Anagnostakis • Panagiotis Trimintzios
Editors

Proceedings of the 3rd European Conference on Computer Network Defense

Springer

Editors

Vasilios Siris
Foundation for Research &
Technology, Hellas
Inst. Computer Science
PO Box 1385
711 10 IRAKLION, CRETE
GREECE

Kostas Anagnostakis
Foundation for Research &
Technology, Hellas
Inst. Computer Science
PO Box 1385
711 10 IRAKLION, CRETE
GREECE

Sotiris Ioannidis
Foundation for Research &
Technology, Hellas
Inst. Computer Science
PO Box 1385
711 10 IRAKLION, CRETE
GREECE

Panagiotis Trimintzios
European Network &
Information Security Agency
(ENISA)
PO Box 1309
710 01 IRAKLION, CRETE
GREECE

ISSN: 1876-1100 e-ISSN: 1876-1119
ISBN: 978-0-387-85554-7 e-ISBN: 978-0-387-85555-4
DOI: 10.1007/978-0-387-85555-4

Library of Congress Control Number: 2008933668

Preface

The 3rd European Conference on Computer Network Defense took place in September 2007 at Aldemar Hotel, in Heraklion, Crete, Greece in cooperation with the European Network and Information Security Agency (ENISA).

The theme of the conference was the protection of computer networks. The conference drew participants from academia and industry in Europe and beyond to discuss hot topics in applied network and systems security.

The conference was a great success, with 6 refereed papers and 6 invited presentations on topics ranging from high assurance networks of virtual machines to signaling vulnerabilities in wiretapping systems.

This book contains the refereed as well as refereed papers. We are greatful to the authors and presenters for their contributions, as well as the participants of EC2N'07 for making the conference a success.

We are looking forward to a successful EC2ND event in 2008.

K. G. Anagnostakis, S. Ioannidis, V. Siris

Contents

1
Tales from the Crypt: Fingerprinting Attacks on Encrypted Channels by Way of Retainting

Michael Valkering, Asia Slowinska, and Herbert Bos

Department of Computer Science, Vrije Universiteit Amsterdam
Amsterdam, Netherlands
{mjvalker, asia, herbertb}@few.vu.nl

Abstract. Paradoxically, encryption makes it hard to detect, fingerprint and stop exploits. We describe Hassle, a honeypot capable of detecting and fingerprinting monomorphic and polymorphic attacks on encrypted channels. It uses dynamic taint analysis in an emulator to detect attacks, and it tags each tainted byte in memory with a pointer to its origin in the corresponding network trace. Upon detecting an attack, we correlate tainted memory blocks with the network trace to generate various types of signature. As correlation with encrypted data is difficult, we retaint data on encrypted connections, making tags point to decrypted data instead.

1 Introduction

Intended to enhance the security of network communication, encryption also makes it harder to detect and analyse attacks on the Internet. Strong encryption and pacing on network links lead to traffic that is more or less uniformly distributed in space and time, preventing the extraction of useful information. Methods relying on the observation of traffic characteristics no longer work. Examples include Snort and Bro that use byte patterns [15, 20], analysis of executable code in the network [17], static analysis techniques [8, 23] when applied in the network, and analysis of protocol fields [11, 12]. In addition, while advanced honeypot systems like Vigilante [3], TaintCheck [14] and Argos [18] would *detect* attacks, most

V. Siris et al. (eds.), *Proceedings of the 3rd European Conference on Computer Network Defense*, Lecture Notes in Electrical Engineering,
DOI 10.1007/978-0-387-85555-4_1, © Springer Science+Business Media, LLC 2009

common techniques for signature generation cannot be directly applied. Paradoxically, the very nature of encryption may turn against the original security goals.

At the same time, the use of encryption is increasing in almost all network services, including file systems, web servers, VPNs, databases, p2p, instant messaging, etc. Rather than considering the fairly narrow set of exploits against encryption libraries themselves (like Linux' Slapper [16], and Windows' SSL Bombs [1]), this work is motivated by the larger class of attacks that exploit the applications *using* encryption. It is well-known that given a choice between port 80 (http) and port 443 (https), attackers tend to opt for 443 almost without exception [19]. The reason is that the content of these channels cannot be so easily inspected by firewalls and virus scanners.

Instead of providing a NIDS with copies of the servers' private keys [13], something administrators may be reluctant to do, we prefer to push fingerprinting to the end-application on the host. On the other hand, we do not want to code manually a specific solution for each application. Rather, we are interested in methods for signature generation that apply to a wide variety of applications.

We emphasise that channel encryption should not be confused with polymorphism. Even though encryption yields unique network appearances for all network attacks, the nature of the attacks (polymorphic or not) still surfaces after decryption.

This paper discusses *Hassle*, a honeypot capable of detecting and fingerprinting monomorphic and polymorphic attacks on SSL-encrypted channels. *Hassle* is not application-specific and can be applied to any process that uses SSL for secure communication. We discuss both its design and its implementation on an x86 Linux-based architecture.

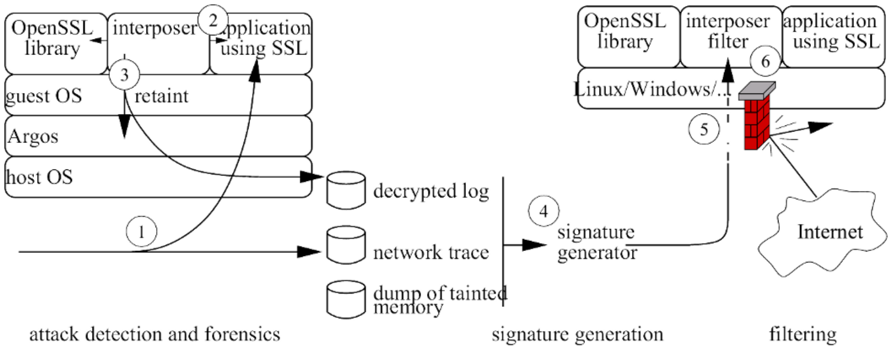


Fig. 1. Interposition, retainting, and signature generation.

SSL encryption. Encryption can be applied at many layers in the protocol stack. The most common examples in practice include the data-link layer (WEP, WPA), the network layer (IPSec), and the application (SSL). As layer-2 encryption in the NIC reduces the problem to that of non-encrypted channels at the OS level and can therefore be handled easily by emulators with dynamic taint [18], the most interesting design alternatives to consider in practice concern IPSec and SSL. Without loss of generality, we opted for implementing *Hassle* for SSL, as it is supported by many servers. Nevertheless, the same techniques can be applied at other layers.

Contribution. To the best of our knowledge, we are the first to address the problem of signature generation (and attack filtering) for encrypted communication, while also handling non-encrypted channels. The techniques we describe are applicable to most types of encryption and require no modification of the applications that need protection. In addition, we describe various novel signature generators that combine with *Hassle*. Besides well-known Snort-like signatures [20], we generate very accurate signatures for *polymorphic* buffer overflows on heap or stack.

The remainder of this paper is organised as follows. Sections 2 and 3 discuss architecture and implementation, respectively. *Hassle* is evaluated in Section 5. In Section 6, we discuss related work, while conclusions are drawn in Section 7.

2 Architecture

At the highest level, our system consists of a detection engine, a signature generator, and a filter, as illustrated in Fig. 1. The detector is a honeypot based on a full-system hardware emulator that provides taint analysis. For this purpose, we modified an existing honeypot, known as Argos [18].

Assume for now that no encryption is used. All data from the network is logged to a rolling trace file (1). By means of taint analysis, *Hassle* tags and then tracks network data throughout the system (1), where a tag points to the origin of the data in the network trace. Whenever tainted data is used in a way that violates the security policy, we raise an alarm. Examples of such behavior include attempts to execute tainted data. At that point, *Hassle* dumps as much relevant data to disk as possible. For instance, for the process or kernel under attack, we save all tainted memory blocks with their tags, the names of the executable and the libraries used, and the address that triggered the alert together with its origin.

The signature generator correlates the data dumped by *Hassle* with those of the network trace to determine a signature (4). For instance, one of our signature generators dissects the input stream to determine which protocol fields were responsible for a buffer overflow and computes an upperbound on the combined length of the protocol fields as a signature. Any message in which the length of these protocol fields exceeds the upperbound is guaranteed to result in an overflow. We use the signature to block the attack elsewhere in the network without needing heavy-weight instrumentation (5).

Unfortunately, in case of encryption correlation between network trace and memory dump is not possible as all memory tags point to seemingly meaningless, uniformly distributed bytes in the network trace (2).To solve this we want to restore a meaningful correlation, albeit not to the network trace directly. For encrypted channels we *retaint* the tagged data after decryption (3). Concretely, we use library interposition to place a small amount of code between the application and the encryption library. The interposer requests the emulator to retaint data using offsets in the decrypted streams as tags. The decrypted data is stored in a log for future use. Although interposing leads to exposure of private data, we regard this (confined) exposure as less threatening than the possibility that the system is completely compromised by an attack via that data.

Signature generation (4) now progresses much like that of non-encrypted channels, albeit at a higher level in the protocol stack. When working at this level, well above the transport layer, we cannot simply dissect the network data stream from the first network packet onwards to determine the protocol fields that were used in the attack. Instead, we use the retainted decrypted network stream. Similarly, the filters that block traffic that contains the signature also must operate at higher-level protocol units (6). We implement them as *interposer filters* between the SSL library and the application that flag or drop all traffic towards the application that matches the signature.

Before we start discussing the precise nature of our signatures, we mention that as much as possible we focus on exploits rather than payloads, for several reasons. First, exploits exhibit fewer opportunities for polymorphism. Second, one exploit is often used for different payloads and stopping it kills multiple birds with one stone. Third, blocking exploits prevents many attacks from entering the system altogether.

2.1 Tracking Issues

Tagging and tracking conceptually consists of adding meta-data to every byte in memory. The more meta-data is added, the more powerful the attack analysis can be. Again, we first consider the case that no encryption is used, and subsequently address encrypted channels in Section 2.2.

In *Hassle*, we allow three types of tagging. The cheapest, (incurring an overhead of approximately 15× on average), but also the weakest, is known as *black-white tagging* and simply indicates for each byte whether it originated in the network. It provides no clues as to the origins of tainted data in memory other than that it came from a suspect source. The tag is a single bit and no immediate correlation of network trace and memory is possible. It may be possible to align patterns in memory and network trace by means of similarity search [11, 18], but the margin of error is large.

A more powerful tagging method is known as *net tracking*, which keeps track of the network origin of tainted memory. In *Hassle*, we have implemented two modes of net tracking: *full origin*, and *single origin*.

Full-origin tracking is the most precise form of net tracking, but also the slowest (with a slowdown of about two orders of magnitude). Whenever data arrives from the network we tag it with a pointer to the corresponding bytes in the network trace. Whenever two tainted values are combined (e.g., an addition of two tainted locations), we retain the tags of both of them. This is implemented by maintaining a set of origin pointers that refer to the tags of the (one, two or three) tainted operands that produced this data. By applying the procedure recursively whenever tainted values are combined, we construct a tree with leaves pointing into the actual network trace. In practice, the amount of memory needed for the administration is approximately three origin pointers per tainted word. The overhead of maintaining the origin pointers is also considerable.

In contrast, single-origin tracking retains a single origin pointer that points to a byte in the network trace directly. If two tainted values are combined, we pick one of the tags for the destination. Single-origin tracking introduces some imprecision in the tracking. In practice, however, we have not seen instances where such imprecisely tagged data can be exploited by attackers. The advantage of single-origin is that it is much more efficient both in memory (one word per word of tainted data) and in computation (reducing the overhead to less than 20×). For this reason, we have used single-origin tracking for this paper. If the nature of applications changes such that single-origin tracking becomes an issue, we can switch to full-origin tracking in the future.

The strongest tagging method, known as *age-stamped net tracking*, maintains not only (full or single origin) net tracking, but also age stamps per tainted value. The age stamps serve to separate different buffers on the stack or the heap. For instance, every function call results in a new age stamp, and all tainted stores in the function are associated with that age stamp. As a result, it is easy to separate the heap or stack data contributing to the attack from stale tainted data left by a previous function frame.

Besides age stamps, this tagging method tracks a small amount of additional meta-data. For instance, it inserts red markers just above and below a buffer allocated on the heap. An overflow of this buffer triggers a reaction in the emulator (e.g., to log the buffer contents for later correlation). In addition, we maintain two bits per tainted byte to distinguish between different overlapping tainted buffers with the same age stamp.

The details and analysis of age stamps and related meta-data is quite complex and beyond the scope of this paper. Interested readers are referred to [22]. What is important for this paper is that when a buffer overflow attack is detected, the origin pointers, age stamps and additional meta-data combined allow us to determine with great accuracy the exact bytes that contributed to an overflow. The overhead of age stamps in memory consists of an additional word per tainted address. The computational overhead is modest, less than 20% compared to single-origin tracking without age stamps in real applications like Apache.

In principle, any tagging method can be used for *Hassle*. However, as alignment is error prone in black-white tagging, we mostly used single-origin tracking for our experiments, and age-stamp tracking where indicated.

2.2 Retainting

Regardless of tagging method, origin pointers are useless in the case of encrypted channels. The reason is that without the key we cannot perform the one-to-one mapping between bytes in memory and bytes in the network trace.

For this reason, *Hassle retaints* all encrypted data. Rather than pointing to a specific byte in the encrypted network trace, we make it point to a specific byte in the *decrypted* SSL stream. Of course, the nature of decrypted streams is different from that of the network trace. For instance, layer 2–4 headers are not visible and TCP flows have already been reassembled. As a result, we will have to adjust the signature generation and filtering components accordingly.

Two implementation issues remain. First, after separating encrypted and non-encrypted data we must retaint the data right after decryption in such a way that a tag used for retainting is unique across all streams. As a result, the tag cannot be a simple offset into any one particular SSL stream. Second, we should be able to uniquely identify SSL conversations and associate incoming data with an SSL stream. In the next two sections, we discuss our solution to each problem separately.

2.2.1 Determining the Tag

As decryption occurs in user space, we employ light-weight interposing libraries between the application and the SSL functions. Whenever a read is performed on an SSL stream, the data will be decrypted. At that point the interposer requests a retaint for the decrypted data and logs the decrypted data to file. Beyond that, the interposer serves as a low-overhead relay between the SSL library and the application.

While the interposer trivially knows the offset of decrypted data in the corresponding SSL stream, determination of the tag should not take place there. Given a tainted data item, *Hassle* must be able to identify exactly the decrypted SSL block in which it originates. In other words, a tag must be unique not only within its own SSL stream, but across all streams. Doing such retainting in the interposer requires adding a unique SSL stream identifier to each tag, which is both complex and expensive in memory.

Instead, we perform trivial retainting in the emulator and push all complexity to detection time. For the remainder of this section, refer to Fig. 2 which zooms in on the decrypted data log and shows a situation where three SSL connections are active; the decrypted data blocks in the channels are tagged by the emulator.

We maintain, conceptually, a single log for all SSL streams and let the emulator determine a tag consisting of an offset in this global log. In reality, we store each SSL stream in separate append-only logs identified by a unique SSL stream identifier (the nature of which will be discussed in Section 2.2.2). For instance, the tags in Fig. 2 refer to an offset in the global input. That is, the blocks that are tagged 0, 10, and 20 indicate that the first block starts at global offset 0, and since the next block starts at offset 10, the first block contains 10 bytes. Similarly, the third block starts at offset 20, so the second block also contains 10 bytes. However, while this is the second block in the global input, it is the first block in SSL stream 2. For completeness, the figure also shows on the left some tainted data that has been copied, leading to tag propagation.

In other words, *Hassle* orders and tracks all updates to the decrypted data log in a global order, layering a virtual append-only global log over the individual SSL stream logs. The log for SSL stream 1 in Fig. 2 contains two data blocks, containing 10 and 20 bytes respectively. The global log, on the other hand, consists of five data blocks. Blocks 1 and 2 both contain 10 bytes; block 3 contains 50, and so on. *Hassle* tags the decrypted

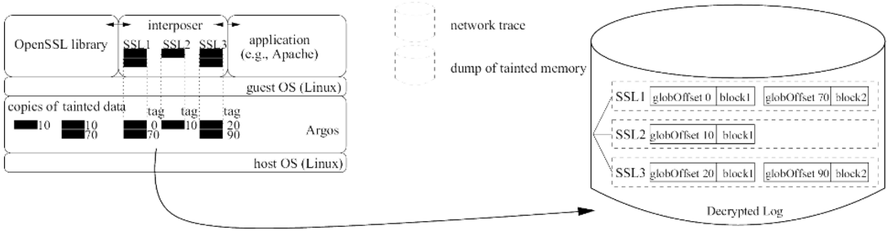


Fig. 2. Global offsets in the decrypted data log.

data with an offset into the global log, trivially guaranteeing uniqueness. When an attack is detected, the tags of offending bytes point to a specific block in the global log. We maintain a simple index to find the corresponding SSL stream and hence all decrypted data.

Finally, for each SSL stream we also store the original tag of the first decrypted data block. This tag points to a byte in the encrypted network trace where it originated. Thus, we are always able to find the network flow that carried the attack, which in turn enables us to identify the IP addresses and port numbers of the attack.

In summary, for retainting the interposer asks the emulator to determine a new tag for the data as explained earlier. It then pushes the decrypted data to the decrypted data log. Currently, this is implemented as a request over a UDP connection to the host OS. As UDP is unreliable, we take into account potential reordering and loss. For the first chunk of decrypted data in the SSL stream, we also log the association between the decrypted data and the original tag, enabling us to recover conveniently the network flow (IP addresses, ports, etc.) in which an attack originated.

The administration of all other meta-data works in exactly the same fashion as in the non-encrypted version. In particular, this is true for the meta-data that is kept for age stamp tracking, such as age stamps and red markers as described in Section 2.1.

2.2.2 Identifying the SSL Conversation

The construction of a unique identifier for a single conversation is not trivial. Ideally, the identifier should be a unique number derived from one or more fields of the SSL connection structure. Simply using the memory address of the `ssl` structure (see Listing 1) will not suffice, because new conversations may reuse the structures associated with old conversations.

However, the handshake phase of SSL (version 3) connections includes the exchange of unique challenges by client and server, which can be obtained from the SSL structure. Unlike client challenges, server challenges cannot be influenced by clients, and are thus well-suited for identifying the conversation. Unfortunately, SSL version 2 does not support server challenges. While older versions of SSL are not our main concern, we decided to add some support for version 2. For such conversations, we currently resort to a combination of the client challenge with the memory address of the SSL connection structure and the thread id of the process using the OpenSSL library. Admittedly a hack, the values of the latter two are not controllable by any attacker and the combination is pseudo-unique.

2.3 Interposition Details

SSL conversations start with a handshake phase that deals with authentication and creation of a session key. No application data is transmitted during this phase and we therefore do not monitor it. This phase also creates the SSL structure for the conversation.

Whenever an application calls `SSL_read` to decrypt and read data, we intercept the call by way of library interposition. Besides the call to `SSL_read`, we are interested in a small subset of other calls, including `SSL_shutdown` and `CRYPTO_num_locks` and a few others[1]. As an example, we show the code for the `SSL_read` interposer in Listing 1. In the first few lines we find (line 2) and execute (line 3) the real `SSL_read` function as requested by the client. Next, we retaint the data and log the decrypted stream using the `retaint_netidx` and `inform_logclient` functions, respectively. None of the retaint functions are visible to the caller, rendering the interposer transparent to the client.

Listing 1: Interposer for `ssl_read` library function

```
01 int SSL_read(SSL *ssl, void *buffer, int length) {
02   int(*func)() = (int(*)())dlsym(RTLD_NEXT,"SSL_read");
03   int func_result = func(ssl, buffer, length);
04   retaint_netidx(...); // now retaint
05   inform_log(...);     // log decrypted data (Fig. 1)
06   return func_result;  // return original result
07 }
```

A call to `SSL_shutdown` simply leads to destruction of state maintained by *Hassle*. `CRYPTO_num_locks` is more complex. OpenSSL uses a number of global data structures that will be implicitly shared when multiple threads use the library. To use the library in the context of threads safely, we need locks to prevent simultaneous access to the global structures and `CRYPTO_num_locks` returns the number of locks needed by the library to synchronise access. Because our interposing library also implements a global data structure, the number of locks should be increased by one. So we interpose this function to make another lock available to protect the global data structure.

3 Signature Generation

As illustrated in Fig. 1, signature generation is devolved from detection and different generators can be plugged into the architecture. We currently

[1] In fact, we interpose SSL_write also, but the reasons for doing so are related to attack replaying and beyond the scope of this paper.

support two main classes of generator that will be referred to as *pattern-based* and *vulnerability-based*, respectively. Pattern-based signatures are widely used in network intrusion detection systems such as Snort [20] and Bro [15]. They consist of a basic identification of the type of packet (e.g., TCP or UDP and port number), together with a byte pattern which is matched against traffic of the appropriate type.

In vulnerability-based signatures we focus on buffer overflows on the heap and stack and decouple the signature from the attack's content in bytes completely. The signature consist of a bound on the combined length of a set of protocol fields. Any message where such fields have a combined length that exceeds this bound will incur an overflow, regardless of their content, so these signatures cater well to polymorphic attacks. On the surface, they are quite similar to those of Covers [11], but we will show that they are considerably more accurate.

Each class of signatures has four variants of generators: (1) encrypted vs. non-encrypted, and (2) single-origin net tracking with age stamps vs. single-origin net tracking without age-stamps. The main difference between encrypted and non-encrypted channels, as far as signatures are concerned, is where they are applied. For non-encrypted channels, we are able to apply signatures in the network before the malicious traffic reaches the host (indicated by (5) in Fig. 1). In contrast, encrypted channels require the filters to be applied at a higher level, i.e., as an *interposer filter* in user-space (indicated by (6)). In addition, an interposer filter must know which signatures to apply. To do so, the signature generators consults the network tag that was stored in the decrypted data log to find the corresponding flow in the network trace. By means of the flow, we obtain the port numbers used in the attack (and possibly other network-specific information). Finally, the forensics data generated by *Hassle* specifies details about the application under attack. This is then used by remote clients to determine which interposer filters should apply the filter.

3.1 Pattern-Based Signatures

Our pattern-based signatures handle all buffer overflows and all code injection attacks (with slightly better signatures for buffer overflows). Note that while code injection may be caused by buffer overflows, there also exists exploits for double frees, format strings, etc. Regardless of exploit, *Hassle* is able to fingerprint such attacks also. We distinguish between signature generators with and without age-stamp analysis.

*Single origin net tracking without age stamp analysis (***SontNoAsa***).* Whenever *Hassle* detects an attack, we determine whether the program counter (EIP) register was tainted. If so, we use the origin pointer of the register to locate the corresponding byte in the network trace. Next, we perform a correspondence search between the flow content in the network trace prior to this byte and the tainted data in memory. Whenever the tags point to the appropriate values in the trace, we include them in the pattern. Pseudo-code for this naïve algorithm is shown in Listing 2.

Listing 2: Naïve generator for pattern-based signatures
```
char *addr = top of memory location loaded in EIP;
sig_t sig = { *addr }; // signature as byte sequence
while (tag(addr-1) == tag(addr)-1)
    sig = concatenate (*--addr, sig );
```

Hassle improves on this naive scheme by taking into account simple gaps in the tainted memory region and Unicode character encodings (e.g., UTF-8). Gaps may occur in tainted buffers for many reasons, e.g., due to non-tainted assignments to overflown memory *after* the buffer overflow occurred and *before* the control flow was diverted. For instance, consider the (contrived) code snippet in Listing 3.

Listing 3: Tainted data: gaps in tainted data
```
01 void read_from_socket (int fd) {
02   int n;
03   char unrelated_1 [8];     // not vulnerable
04   char vuln_buf [8];        // vulnerable buffer
05   char unrelated_2 [8];     // not vulnerable
06   read (vuln_buf, fd, 32);  // overflow
07   read (unrelated_1, fd, 8);// tainted gap
08   read (unrelated_2, fd, 8);// adjacent buffer tainted
09   n = 1;                    // untaints 4 bytes: gap
10   return;
11 }
```

While the code is not very realistic, it serves to illustrate a number of complications that prevent the naive solution from producing correct results in *some* cases. Before the attack is detected (when `return` is executed), the assignment in line (9) creates an untainted gap of 4 bytes in the tainted buffer. Similarly, the read in line (7) creates a tainted gap filled with unrelated data. Finally, the vulnerable buffer may adjoin another buffer that also contains tainted data, as demonstrated by the read in line (8).

In the pattern-based signatures generated using SontNoAsa, gaps are detected by looking at the tags. Gaps either have no tags, or tags with unexpected values. Gaps are skipped whenever the byte on the other side of the gap has the appropriate (expected) tag value. In case there is no such byte, the signature stops here. Unfortunately, the adjacent buffer `unrelated_2` that also contains tainted data cannot be distinguished from the vulnerable buffer and is therefore also included in the signature. As a result, SontNoAsa and pattern-based signature can lead to false negatives.

Hassle can easily cater to well-known forms of encoding like Unicode. Sometimes a network trace carries ASCII data, which is translated to a Unicode representation in memory, or *vice versa.* Either way, the skew between network trace and memory is predictable. As both gap- and Unicode handling are trivial extensions to the naive algorithm in Listing 2 we will not show them here.

*Single-origin net tracking with age stamp analysis (***SontAsa***).* Applying age-stamp analysis improves the accuracy of pattern-based signatures in the case of buffer overflows attacks on heap or stack. Extended age-stamp analysis directly yields all bytes in memory that were used in the overflow. By means of single-origin net tracking we obtain the corresponding network bytes. As we only identify the relevant bytes, compensating for gaps and Unicode is no longer necessary, as it is handled automatically. In fact, the signature generation is considerably more accurate, as age stamp analysis is also capable of distinguishing buffers `vuln_buf` and `unrelated_2`[2].

Code injection signatures. Not all attacks are overflows. Perhaps other means were used to divert control to the code injected by the attacker. Since such code is tainted by nature, the attack is detected when instructions in the tainted region are executed. On rare occasions, injected code may be executed because of legitimate control flow (i.e., a *bona fide* jump to a memory area that is tainted). More commonly, the jump is the result of a control flow diversion by means of a *format string* attack, *heap corruption*, or the *overflows* mentioned earlier. Regardless of how the injected code is reached, we again align memory and network trace to generate a signature. The only difference is that if the attack is not an overflow, we match against the injected code.

In case we detect both a buffer overflow and code injection (i.e., the buffer overflow was used to divert control to the injected code), we have to decide whether to use as signature either the match against the injected code, or the match against the overflow bytes. As injected code is more likely to be polymorphic than the exploit itself, we favour the overflow signature. As a rule of thumb we use the injected code signature only if (a) it is longer than the overflow signature, and (b) if the length of the overflow signature is less than some minimum length L_{min} (e.g., L_{min}= = 12).

Limitations. Pattern-based signatures are attractive because of their simplicity, and their popularity in existing IDSs. Unfortunately, they are also fairly weak and incur both false positives and false negatives. In particular, by using the actual content of the attack, traffic pattern-based signatures

[2] And indeed more complicated cases. For details, see [22].

are powerless against polymorphic attacks. They also do not work when multiple tainted buffers are adjacent in memory.

3.2 Signatures for Polymorphic Buffer Overflows

For polymorphic buffer overflow attacks we decouple the signature from an attack's content in bytes. Instead, we look at the *vulnerabilities*. A message that causes a buffer overflow contains one or more protocol fields of unusual length that, when copied collectively into a vulnerable buffer, overwrite critical data. Vulnerability-based signatures establish a maximum length *L* for the field(s). Any message where the combined length of these fields exceeds *L* is sure to overflow the buffer. We first discuss a naive implementation that is also used by other projects and demonstrate why it is flawed. Next, we explain how age stamp analysis helps us solve the problems.

*Single-origin net tracking without age-stamps (***SontNoAsa***)*. In this naive implementation, we trace the point of attack *X* to a byte *N* in the network trace and establish what protocol field *P* contains this byte. Doing so is trivial if traffic is not encrypted. In our case, we reassemble the TCP stream and dissect the higher-layer protocols with a protocol dissector (we use a modified version of Ethereal [2]). After locating the protocol field containing *N* we generate a signature consisting of an identification of the stream and application (port numbers, executable name) together with a bound *L* on the length *P*, where *L* is (*N* – start of *P*). It is likely that any message with *P* longer or equal to *L* results in an overflow (but not certain, as we shall see shortly).

For encrypted traffic the procedure is a little more complicated as we cannot start from the network packets to dissect the input stream. As we start dissecting above the transport layer, how do we decide which protocol dissector to use? We identify three solutions for dealing with the problem. First, we may use custom interposers for specific applications. For instance, we can apply an HTTP interposer for Apache which always assumes HTTP traffic. Second, we may use the port numbers in the network trace as an indication of the protocol (e.g., all port80 traffic will be assumed to be HTTP). Third, we may use the information about the application as an indicator for the protocol (if the application is "apache", we use the HTTP dissector). Currently, we use hard-coded associations.

In our implementation, we modified the Ethereal protocol analyser [2] to start from higher-level protocols and to work with incomplete protocol messages. As this signature generator is very similar to Covers [11], we refer to it as *Hassle*-Covers.

Limitations. Unfortunately, Covers (and thus *Hassle*-Covers) yields both false positives and false negatives. First, exploits like Apache-Knacker [21] use the fact that sometimes multiple protocol fields are copied in the same buffer to generate an overflow of the buffer with the content from all these fields. As a result, establishing a bound on the length of a single field may miss attacks where the length of *P* is small, but the combined length of all fields exceeds the length the buffer. Similarly, it may misdiagnose a message as malicious when *P* is longer than *L*, even though the combined length of all the relevant fields is less than the buffer size.

The second reason is related, but more subtle. It is also more serious. The dissector used to generate signatures may work at different protocol field granularities than the application itself. For instance, the dissector may identify subfields in a record-like protocol field as separate fields, while the application simply treats it a single protocol field. As a consequence, the two types of misclassification described above may occur even if the exploit does not explicitly use multiple fields. As we generally do not have the application's source code, and hence have no knowledge about the granularity of the application's dissector, this is a serious problem.

*Single-origin net tracking with age-stamp analysis (***SontAsa***)*. To deal with this problem we take into account *all* bytes used in the overflow. A reliable way of establishing which bytes were used in the exploit is by means of age-stamped net tracking. In the case of non-encrypted traffic we find those bytes in the network trace directly. In the case of encrypted traffic those bytes are found in the decrypted data log using the modified Ethereal protocol dissector, as described in the previous sections.

To be precise, we find accurately *all* bytes that were used in the overflow and we do so before a single instruction of the attack is executed and without the need to replay the attack. Gaps in the buffer overflow (as explained in Section 3.1), be they tainted or non-tainted, are duly skipped, and adjacent buffers that are both tainted but different are separated. In addition, encodings like Unicode are automatically handled. The details are complex and beyond the scope of this paper. The exact procedure is explained in [22].

Given the overflow bytes, we then identify *all* protocol fields that were used in the attack and establish an upperbound *L* on their combined length *according to our dissector*. Whether or not the application uses a different protocol field granularity is now immaterial.

4 Filters

Hassle filters for non-encrypted traffic consist of simple checks, either matching pattern-based signatures against network packets, or looking at the length of fields of specific protocol messages for vulnerability-based signatures. They can be applied in the network or in the operating system kernel.

For encrypted channels, similar procedures are used, except that they are applied by means of library interposition in user-space. Filters should only apply those signatures that apply to the specific application that uses the SSL library. Currently, this is done by explicitly associating a separate interposer filter library with every application we want to protect. Each interposer filter only picks up the signatures for the application it is protecting. Since *Hassle* provides the full name of the executable as part of the signatures, the association is trivial. For vulnerability-based signatures, the interposer filters again use the modified version of Ethereal for protocol dissection.

5 Results

For realistic performance measurements we compare the speed of code running on *Hassle* with that of code running without emulation. While this is an honest way of showing the slowdown incurred by our system, it is not necessarily the most relevant measure, as we use *Hassle* as a honeypot rather than a desktop machine. To our knowledge, no automated attacks exist that shun slow hosts, because they might be honeypots.

It should also be mentioned that encryption is known to be one of the most challenging applications for dynamic taint analysis, because decryption requires a large number of tainted operations. For instance, recent work on demand emulation [7] describes a technique to speed up emulation-based taint analysis by switching to fast VM-mode when possible. While many applications incurred as little as a factor 2 slowdown, SSL incurred a slowdown of 150.

Performance. To quantify the observed slowdown we used the Apache 2.2.3 web server using the OpenSSL library. The first simple test consisted of requests to read a 5 MB block from the client to the server, which on top of a vanilla Qemu emulator took Apache 19.9s to complete (2.06Mbps). On *Hassle*, the same task took 23.47 s (1.75 Mbps), incurring a 15% overhead.

We also evaluated Apache throughput in terms of number of processed requests per second and the corresponding average response time. We used `httperf`[3] for generating requests. The experiments were conducted on a dual Intel™ Xeon at 2.80 GHz with 2 MB of L2 cache and 4 GB of RAM. The system was running SlackWare Linux 10.2 with kernel 2.6.15.4.

[3] www.hpl.hp.com/research/linux/httperf/.

Table 1. Maximum rates for https connections.

Description	Average (req/s)	Standard deviation	Relative to native	Response time (ms)
https/native	57.0	0.3	1.0	21
https/Argos	0.6	0.07	95.0	87
https/Hassle	0.55	0.12	103.6	63
http/Argos	38	1.8	n/a	147
http/Hassle	38	1.7	n/a	200

The results for https (using SSL) are summarised in Table 1. The table lists results for 3 Apache configurations: (i) running natively, (ii) running on the Argos honeypot, and (iii) running on *Hassle*. We also show some results for non-encrypted http connections for comparison[4]. The results are the best possible in the sense that at this rate the webserver was able to keep up fully with the request rate, while not yet incurring unreasonably long response times. For instance, for all reported rates the response times were below 200 ms. Beyond these rates, response times shot up to many hundreds or even thousands of milliseconds.

The experiments confirm that SSL is very expensive for dynamic taint analysis, incurring a slowdown of approximately a factor 100 over native code running SSL, and a factor 70 over non-encrypted channels using the same (emulated) configuration. Consequently, dynamic taint analysis for SSL encrypted channels is only viable on honeypots, and even here the number of connections should be limited. Note however, that slowness is not really a major issue for a honeypot as long as it is able to serve a request sufficiently fast. Moreover, in most deployments of honeypots like Argos (e.g., at SURFnet[5], Eurecom [10], and in the Noah project[6]), a first-pass filter of low-interaction honeypots is used to shield the high-interaction honeypot from most requests. The second thing to observe is that there is little difference between *Hassle* and the original Argos (i.e., a honeypot without retainting and logging of encrypted data).

Probing further, it appeared that most of the overhead is in the connection set-up where SSL uses asymmetric encryption. As a result, performance improves significantly when use is made of https sessions. For instance, for 100 sessions per connection, the reply rates for https *Hassle* are shown in Table 2.

[4] We were unable to measure reliably the native version for plain http, because httperf at the client side became the bottleneck.

[5] http://honey.surfnet.nl/

[6] http://www.fp6-noah.org/

Table 2. Maximum rates for https connections with sessions.

Description	Average rate (req/s)	Relative to native
https/native with sessions	486	1.0
https/Argos with sessions	31.1	15.6
https/Hassle with sessions	25.0	19.4

Micro-benchmarks. Retainting itself is not very expensive. We measured 200µs on a Pentium M at 1.4GHz with 1GB of RAM, running Ubuntu Linux 6.0.6. The guest OS ran Ubuntu Linux 5.05 with kernel 2.6.12.9, on top of Qemu 0.8, Argos and *Hassle*. Similarly, the overhead of the entire interposition library to do the retainting is modest. We measured performance with and without the interposition library for both SSL reads and SSL writes for block sizes ranging from 100B to 16 kB bytes. For writes, the relative overhead lies between 17.5% for the largest blocks and 26% for the smallest ones. For reads, the results range from 24.5% for the largest to 50% for the smallest blocks. Likewise, in our evaluation the interposer filters that scan SSL streams for the occurrence of a signature incurred overheads between 2 and 15% compared to a system without the filters, for the most expensive (pattern-based) signatures.

Generating a single-origin net tracking signature. In a cursory evaluation of the signature generator we tested the generators for pattern-based and vulnerability-based signatures using request sizes of 170 B and 100 kB, respectively (representing different amounts of data to dissect and/or scan). The pattern-based signature was generated in 1.1 ms for the small request, and 14 ms for the large one (median values). Vulnerability based signatures for polymorphic attacks required 3.9 ms, and 26 ms to be completed (assuming the protocol dissector is loaded already).

6 Related Work

To our knowledge, we are the first to tackle the problem of one-shot signature generation for communication on encrypted channels. Dynamic taint analysis, on the other hand, is well-known and used in TaintCheck [14], Vigilante [3], and Argos [18]. None of these projects offer signatures for encrypted traffic.

Library interposition as a way of monitoring interaction with libraries is used frequently to analyse applications [4] and generate audit trails [9]. Liang et al. propose library interposition to learn about program inputs that lead to crashes induced by buffer overflows [12]. In essence, they consider library calls made from a given program context and raise an alert when an input is significantly longer than the maximum input length seen in the past. Interposition is also applied at the system-call level either to confine the application [6], or to monitor the compliance of a sequence of calls with a predefined application model [5, 12]. In contrast, we intercept library calls to switch to tracking decrypted network streams by adjusting the tags in dynamic taint analysis.

Several of our signature generators are based on existing work. In particular, the pattern-based signatures are quite popular in open-source NIDSs like Snort [20] and Bro [15]. However, the way we generate the signatures is a little different from existing projects. This is true for the way *Hassle* skips gaps and handles Unicode, and even more so for the age-stamped net tracker that determines accurately which bytes are used in a buffer overflow.

Similarly, the single-field vulnerability-based signature is already proposed in Covers [11]. We have demonstrated that such signatures have fundamental flaws and shown how we solved them.

Application-level filtering is performed by virus scanners and Vigilante. Filters in interposing libraries are not very common. While the paper is a bit vague about it, we suspect that they are also used in ARBOR [12], although the filters are of a very different nature.

7 Conclusions

We have described *Hassle*, a honeypot system that is capable of generating signatures for communication over both encrypted and non-encrypted channels. For encrypted traffic we retaint the tainted data by making the tags point to the decrypted SSL streams. Different types of signature generator can be used in the system. Which one should be used is a tradeoff between simplicity and accuracy. In our opinion, pattern-based signatures are useful for simple, non-polymorphic attacks, while vulnerability-based signatures work well with more advanced, polymorphic exploits. To our knowledge, we are the first to develop a system capable of fingerprinting attacks over encrypted channels and cater to both monomorphic and polymorphic exploits.

References

[1] P. Bueno. IIS Exploit released / Gagobot.XZ – SANS Microsoft Advisories. http://isc.sans.org/diary.html?date=2004-04-14, April 2004.
[2] G. Combs. Ethereal network protocol analyzer. http://www.ethereal.com.
[3] M. Costa, J. Crowcroft, M. Castro, A Rowstron, L. Zhou, L. Zhang and P. Barham. Vigilante: End-to-end containment of internet worms. In *Proc. of*

the 20th ACM Symposium on Operating Systems Principles, Brighton, UK, 2005.
[4] T. W. Curry. Profiling and tracing dynamic library usage via interposition. In *Usenix ATC*, Boston, MA, June 1994.
[5] J. Giffin, S. Jha, and B. Miller. Efficient context-sensitive intrusion detection. In *11th Annual Network and Distributed Systems Security Symposium*, 2004.
[6] I. Goldberg, D. Wagner, R. Thomas, and E. A. Brewer. A secure environment for untrusted helper applications. In *Proceedings of the 6th Usenix Security Symposium*, 1996.
[7] A. Ho, M. Fetterman, C. Clark, A. Warfield, and S. Hand. Practical taint-based protection using demand emulation. *SIGOPS Oper. Syst. Rev. (Proc. of ACM SIGOPS EuroSys, April 2006, Leuven, Belgium)*, 40(4):29–41, 2006.
[8] C. Kruegel, E. Kirda, D. Mutz, W. Robertson, and G. Vigna. Polymorphic worm detection using structural information of executables. *8th International Symposium on Recent Advances in Intrusion Detection (RAID)*, Sept 2005.
[9] B. A. Kuperman and E. Spafford. Generation of application level data via library interposition. Technical Report CERIAS TR 1999-11, 1999.
[10] C. Leita, M. Dacier, and G. Wicherski. SGNET: a distributed infrastructure to handle zero-day exploits. Technical Report EURECOM+2164, 2007.
[11] Z. Liang and R. Sekar. Fast and automated generation of attack signatures: a basis for building self-protecting servers. *CCS '05*.
[12] Z. Liang, R. Sekar, and D. C. DuVarney. Automatic synthesis of filters to discard buffer overflow attacks: A step towards realizing self-healing systems. In *USENIX Annual Technical Conference - short paper*, Anaheim, CA, 2005.
[13] McAfee. Encrypted threat protection – network IPS for SSL encrypted traffic. White paper, February 2005.
[14] J. Newsome and D. Song. Dynamic taint analysis for automatic detection, analysis, and signature generation of exploits on commodity software. In *Proc. of Network and Distributed System Security Symposium (NDSS)*, 2005.
[15] V. Paxson. Bro: A system for detecting network intruders in real-time. *Computer Networks*, 31:23–24, December 1998.
[16] F. Perriot and P. Szor. An analysis of the slapper worm exploit - white paper. Technical report, Symantec Security Response, 2002.
[17] M. Polychronakis, K. G. Anagnostakis, and E. P. Markatos. Network-level polymorphic shellcode detection using emulation. In R. Büschkes and P. Laskov, editors, *DIMVA*, volume 4064 of *Lecture Notes in Computer Science*.
[18] G. Portokalidis, A. Slowinska, and H. Bos. Argos: An emulator for fingerprinting zero-day attacks. In *Proc. ACM SIGOPS EUROSYS'2006.*
[19] C. Prosise and S. U. Shah. Hackers' tricks to avoid detection. WindowSecurity White Paper, http://secinf.net/info/misc/tricks.html, 2002.
[20] M. Roesch. Snort: Lightweight intrusion detection for networks. In *Proceedings of the 1999 USENIX LISA Systems Adminstration Conference*.

[21] SecurityFocus. Can-2003–0245 apache apr-psprintf memory corruption vulnerability. http://www.securityfocus.com/bid/7723/discussion/, 2003.

[22] A. Slowinska and H. Bos. Prospector: Accurate analysis of heap and stack overflows by means of agestamps. Technical Report IR-CS-031, Vrije Universiteit Amsterdam, June 2007.

[23] T. Toth and C. Kruegel. Accurate buffer overflow detection via abstract payload execution. In *Recent Advances in Intrusion Detection*, 2002.

2
Towards High Assurance Networks of Virtual Machines

Fabrizio Baiardi[1] and Daniele Sgandurra[2]

[1] Polo G. Marconi La Spezia, Università di Pisa, Italy
[2] Dipartimento di Informatica, Università di Pisa, Italy
{baiardi, daniele}@di.unipi.it

Abstract. We propose a methodology to check software integrity based upon virtual machines (VMs) that integrates controls at distinct execution levels. The baseline of the proposed approach is the virtual machine monitor (VMM) capability to access the memory of a VM to apply a set of consistency checks to the VM operating system (OS). In turn, the OS can apply a different set of consistency checks to the application processes, and applications can also enforce a further set of security controls. The union of all the consistency checks forms a chain of trust, where each level controls the integrity of the one above it through the proper interface for that level. In this way, the proposed approach minimizes the semantic gap in-between two different levels, because each level only applies those security controls that are coherent with the view of the level. We apply this methodology to build a distributed intrusion detection system (IDS) to detect attacks against a network of VMs. According to the proposed methodology, the tool adopts VM introspection (VMI) to apply a set of consistency checks to the kernel of the OS of each VM. Then, we extend the kernel of each VM with a set of functions to check the integrity of the processes involved in the detection of intrusions.

1 Introduction

Any approach to intrusion detection should take into account that most sophisticated attacks strive to occupy the lowest system level. This is due to at least two reasons: (i) to achieve a full system control; (ii) to deceive the legitimate system owner by hiding the traces of the compromise and providing the owner with a false sense of security. As an example, *rootkits* have gradually evolved from user-level to kernel-level rootkits [10, 28]. This implies that, since both rootkits and tools that detect them run at the lowest level, the kernel one, only the first one able to predict and prevent the adversary moves can eventually succeed in getting full control of the system [31].

In the context previously described, a fully general, level-independent technique to discover signs of intrusions is *introspection*. This technique analyzes the system memory to rebuild data structures of interest and check their consistency. If no dedicated hardware support is available [26], *virtual machine introspection* [12] can be applied provided that the OS and applications to be monitored are executed inside a VM. A VM is an execution environment created by a virtualization technology that emulates, at software, the behavior of the underlying architecture [7, 34]. To create, manage and monitor the VMs, this technology introduces the *virtual machine monitor* (VMM) [13] a thin software layer in-between the OS layer and the hardware/firmware one. In this way, a standard physical machine supports several VMs, each running a distinct OS. By exploiting the VMM direct access to the memory of each VM, virtual machine introspection can analyze the state of the kernel hosted on a VM. In this way, introspection is applied at a lower level than the one an attacker can gain and, provided that the VMM cannot be subverted, we can build from the VMM up a chain of trust where each level applies a set of different consistency checks to discover attacks. In the following we will use the term introspection rather than the more correct, but longer, virtual machine introspection.

As discussed in the next sections, we believe that VMM and introspection are important building blocks to create high assurance systems. In this view, we have developed *Psyco-Virt*, a software architecture that integrates introspection with a set of host and network IDS tools to achieve high assurance on the integrity of the VMs. The overall architecture consists of a cluster of monitored VMs (Mon-VMs), i.e. the VMs to monitor, and introspection VMs (IVMs) to implement the monitoring. All the Mon-VMs are mapped onto a cluster of physical nodes, and one IVM is introduced for each physical node. All the Mon-VMs are connected by a virtual network, the data one, to exchange application traffic. A further virtual network, the *control network*, connects all the IVMs and each IVM and the Mon-VMs on the same node. This is a private hierarchical network that spans across distinct physical nodes to support the exchange of alerts and introspection information. A set of *IDS agents* on each Mon-VM discovers attempted intrusions/attacks and, in such a case, an agent alerts the IVM through the control network. In Psyco-Virt, the kernel of each Mon-VM guarantees the integrity of the controls implemented by the IDS agents, while an *introspector* running on the IVM exploits the VMM control interface to apply introspection and monitor the kernel of each Mon-VM to discover attacks against the kernel itself. In this way, the overall assurance is achieved in three steps:

1. the IDS agents apply a broad range of security checks, such as file system integrity, denial-of-service detection and prevention, anomaly detection;
2. the kernel of a Mon-VM controls the correct execution and integrity of the IDS agents;
3. the IVM applies introspection to assure the integrity of the kernel inside each Mon-VM.

Obviously, the second step may also use already existing security features, such as those offered by SELinux [21, 20].

The overall architecture provides assurance that critical VM components, such as the kernel and the agents, cannot be tampered with to make them return bogus data. In this way, the *trusted computing base* (TCB) includes all the components protected by the chain of trust starting with introspection. In turn, this strongly reduces the probability of a successful attack against the Mon-VMs. A further advantage of the architecture is that the configuration of each VM can be specialized to minimize the software it runs according to both its role and the applications of interest [33, 27, 4]. Lastly, since IDS agents may be simple wrappers to standard IDS tools, the architecture exploits at best current security tools to discover intrusions and attacks against the Mon-VMs.

The rest of the paper is organized as follows. Section 2 describes the overall architecture of Psyco-Virt and justifies the main assumptions underlying its definition. Section 3 discusses the current implementation and shows some examples of IDS agents and of integrity checks applied at different levels. Section 4 presents an evaluation of security and performance results and describes the current limitations of the prototype. Section 5 discusses related works. Finally, in Section 6 we draw a first set of conclusions and outline future developments.

2 Psyco-Virt Overview

After presenting the overall architecture of Psyco-Virt, we discuss the tasks of the IVMs and of the Mon-VMs.

2.1 Overall Architecture

Psyco-Virt assumes that the applications of interest are mapped onto a cluster of VMs, the Mon-VMs, which are then mapped onto a cluster of physical nodes. Each node runs a VMM and an introspector VM (IVM). In turn, each Mon-VM runs an OS and a set of IDS agents (see Fig. 1). A *collector* on each Mon-VM supports the exchange of alerts and commands among the agents and the IVM. All the Mon-VMs are connected by a data network that supports application and OS traffic. A distinct network, the control one, connects the IVMs and each IVM to the Mon-VMs on the same node. This is a virtual private network, which cannot be accessed from the outside world. The control network connects the Mon-VMs in a hierarchical way to the IVMs to exchange: (i) commands among the IVM and the collector or the agents; (ii) alerts reporting intrusions or attacks from the agents to the IVM; (iii) control information among the IVMs to coordinate the detection.

One of the main tasks of an IVM is to apply introspection to protect the kernel of each Mon-VM. In this way, Psyco-Virt integrates both agents to detect intrusions on the Mon-VMs and introspection to preserve the integrity of the kernel of the Mon-VMs themselves. Each agent is a wrapper for a tool, such as chkrootkit [5] or Snort [30], which monitors critical parts of the system and alerts the IVM through the control network each time an intrusion is suspected. The IVM analyzes the alerts received from the agents and handles the detection of an intrusion or attack by executing a proper action, i.e. stop or freeze the execution of a compromised Mon-VM. The ability of freezing a VM to examine its state in more details is a fundamental advantage and a peculiarity of the virtualization technology.

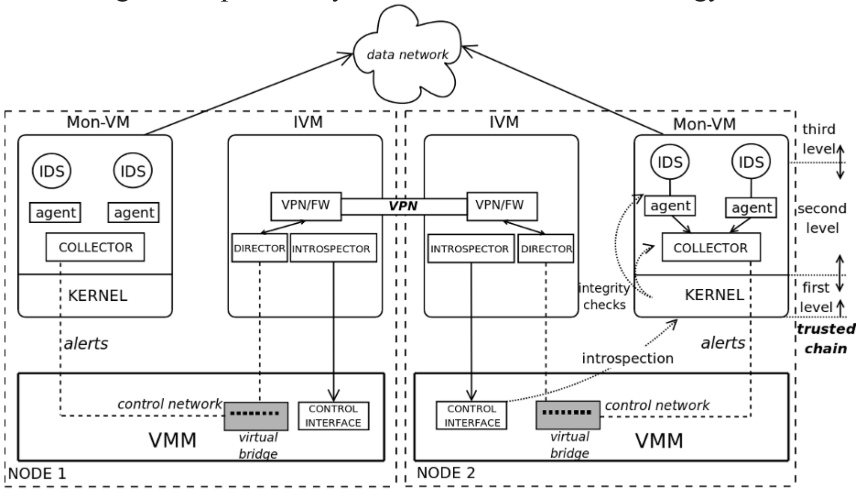


Fig. 1. Psyco-virt Architecture.

Note that a Mon-VM can be introduced just to run an agent, as an example, a network IDS that analyzes the data network traffic. Obviously, the configuration of this Mon-VM should be optimized to reduce the software it runs and hence the opportunity of a successful attack against it.

To justify the multi-level approach we have adopted, consider an alternative solution that applies introspection by modifying an existing IDS tool so that it can apply introspection at the VMM level from the IVM. The complexity of this solution is very high because the IDS tool should monitor the Mon-VMs through a set of checks defined at the hardware level. A further, distinct, approach applies introspection to evaluate a set of consistency checks defined according to the OS-level semantics. This approach does not have to modify the IDS, provided that the introspection library enables the IDS to exploit a high level view of the VMs, defined in terms of files, processes and virtual memory. This is particularly challenging, as in the case of reproducing the structure and the operations of an existing file system. Both approaches are feasible, but the first one requires current IDS tools to be modified so that they work at the hardware level. On the other hand, the second approach requires a complex introspection library that should also support the various versions of the OSes of interest. Our approach is different from both the previous ones. In fact, Psyco-Virt discovers intrusions on a Mon-VM through agents that exploit standard IDS tools, which do not have to be modified, as in the first approach. Furthermore, to simplify the adoption of introspection, our solution extends the kernel with functions that monitor the agents. As a consequence, Psyco-Virt only requires an introspection library much simpler than the one of the second approach because it only evaluates kernel related properties, and the introspector on the IVM can guarantee the integrity of the overall detection system by monitoring the kernel of the Mon-VMs only.

An important assumption of our approach is that the VMM can be trusted and that introspection is the basis of a chain of trust from the VMM to the kernel and then to the IDSes. The reason for assuming that both the VMM and introspection are trusted is twofold. Firstly, the VMM has full visibility of a Mon-VM, because it can access every VM components, such as the memory it allocates to them. Secondly, the VMM is more robust than commodity OSes because: (i) the interface it exports to the higher levels is very simple and so more difficult to subvert than, for example, the one of an OS kernel that implements hundreds of system-calls; (ii) the small size of its code reduces the likelihood of a compromise. In conclusion, since the VMM has full visibility of the VMs it supports and it is essentially isolated from these VMs, the complexity of compromising the VMM or of eluding the introspection monitoring capabilities is very high.

2.2 Introspection VM

The introspector on the IVM applies introspection to discover attacks against critical components of the kernel of a Mon-VM. In particular, it monitors the text section of the kernel and that of the loaded modules to discover whether they have been modified. These memory regions should be read-only, hence any attempt to update them implies that an attacker is trying to insert and execute arbitrary instructions. As an example, this happens when a kernel-level rootkit tries to modify the code of a system call.

When an agent on a Mon-VM detects an attack or when the introspector discovers that the kernel of a Mon-VM has been modified, the IVM handles the corresponding event by executing one of the following actions: (i) stop the execution of a VM and save its state in a file; (ii) kill a process (the offending one); (iii) close the connection of the VM to the data network; (iv) disconnect a user connected to a VM; (v) send an alert to the system console to inform the administrator. Since further information may be required to choose the proper action, the IVM runs a *director* to: (i) collect all the alerts from either the agents through the collector or the introspector; (ii) actively interact with the agents on the Mon-VMs to access specific information about the monitored systems.

The Psyco-Virt introspection library enables the introspector on the IVM to build a high level view of a Mon-VM state starting from the raw data in the Mon-VM memory. This library plays a fundamental role because the VMM control interface provides only a low level view of a Mon-VM, i.e. in terms of memory, registers and disk blocks, whereas the introspector should reason in terms of kernel data structures, such as the process descriptor or the system call table. Thus, the library should translate the low level data received from the VMM control interface into a high level view in terms of kernel data structures. This requires that the library knows the kernel hosted by each Mon-VM, the data structures it uses and the memory areas where they are allocated. In this way, Psyco-Virt can exploit a kernel level view of the status of the Mon-VMs and consider global information such as the list of the running processes, the list of the loaded modules or information associated with a process identifier (PID), such as the list of open files/sockets. This information may be used to implement a very general and effective consistency check: the IVM builds a list of kernel data structures and compares this information against the one returned by an OS command executed on the Mon-VM. Any difference may signal that an attacker is trying to hide her presence.

The control network among the IVMs simplifies the recognition of distributed attacks because the IVMs can broadcast information about attacks, detected either by agents or through introspection. This network is also used to coordinate the shutdown of the Mon-VMs executing applications that are the target of a distributed attack.

2.3 Monitored VM

Each Mon-VM may run several IDS agents, each checking a distinct aspect of the OS or of the applications. The collector receives all the messages from these agents, and immediately forwards any message that reports an attack or an intrusion to the director on the IVM. Custom agents may be created to directly interact with Psyco-Virt to discover intrusions, or existing NIDS/HIDS tools can be used. For example, custom agents could analyze audit logs to discover unsuccessful login attempts. Conversely, tools such as Tripwire [23] may alert the collector any time they detect an attempted intrusion or attack. Each agent behaves like a common host IDS that monitors system calls, audit and applications log files and file system changes. The number of agents and the checks they implement are strongly related to the required security level.

3 Current Prototype

The first prototype of Psyco-Virt has been developed using the C language. The VMM we used is Xen [7] while all the Mon-VMs run the Linux Debian distribution. We also used XenAccess [36] as the basis of the Psyco-Virt introspection library and the Xen Control library to stop a VM, save its state to a file and resume a suspended VM. Lastly, we used OpenSSL [24] and the Linux Cryptolib to compute the hashes, while OpenVPN [25] was used to set up the control network among the IVMs in distinct nodes. The source code of the current prototype is available at http://www.di.unipi.it/~daniele/projects/projects.php.

3.1 Introspection Functions

The following paragraphs discuss the implementation of some sample introspection functions to exemplify some of the capabilities of Psyco-Virt. To implement these functions, we used the headers of the Linux kernel running inside the Mon-VMs, so that through introspection the introspector could reconstruct kernel data structures.

3.1.1 Detecting Kernel Modifications

This introspection function discovers attacks to the kernel of a Mon-VM. The IVM checks the memory pages storing:

- the kernel code, from the address *_text* to *_etext*;
- the system call dispatch table, stored in the *sys_call_table* array;
- the interrupt descriptor table, stored in the *idt_table* table.

Since these pages should never be modified, the introspector periodically computes their hashes to verify that they have not been updated. To detect illegal updates to pages that can be modified, a set of invariants can also be evaluated at runtime [9]. This idea is left for a future work.

3.1.2 Running Processes Checker

According to the general approach previously described, the introspector applies introspection to rebuild the list pointed by the *init_task* symbol and retrieve the set of the running processes on a VM. Then, it compares this set against the one returned by executing the command *ps* on the Mon-VM. If the two sets of PIDs differ, then it is very likely that an attacker has replaced critical system binaries with trojaned versions to hide her presence. In case of systems running a fixed number of processes, more severe checks can be defined because the list of allowed PIDs, paired with the name of the processes, can be fixed during the boot of a Mon-VM.

3.1.3 Loaded Modules Authenticator

This function rebuilds the list pointed by the *modules* symbol to retrieve the list of the modules inserted into the kernel. Then, it checks their integrity and that if they are authorized kernel modules. To this purpose, the first time a Mon-VM is started, we load all the kernel modules it can run. For each module, this function computes the hash of the pages storing its code and save these values and the name of the module in a file. Later, when the Mon-VM is running, this function periodically computes the hash of the pages storing the code of each loaded module and checks if this value differs from the previous one or if the name of a module differs from the stored values. Any difference implies that either a module has been modified or a not authorized one has been loaded. An analogous check is applied to the list of open files.

3.1.4 Promiscuous Mode Checker

This function requests the pages starting from the kernel symbol *dev_base*, a pointer to a list of device structures in the Mon-VM. For each structure, the function verifies whether the corresponding flags indicate that the interface is set into promiscuous mode. This approach applies the same checks implemented in [11], but at a distinct level. In fact, while kstat accesses these structures at the user-level through */dev/kmem* or, if implemented as a module, at the kernel-level, the introspector function of Psyco-Virt applies these checks at the VMM level. Hence, it cannot be defeated even if an attacker gains root privileges.

3.1.5 Anti-Spoofing

To support the anti-spoofing capabilities, the IVM Linux kernel is compiled with the following options:

- CONFIG_NETFILTER_XT_MATCH_PHYSDEV
- CONFIG_BRIDGE_NETFILTER
- CONFIG_NETFILTER_NETLINK
- CONFIG_NETFILTER_XTABLES
- CONFIG_BRIDGE

The IVM implements the anti-spoofing checks on the virtual bridge connecting the VMs on the same node using a set of iptables [22] rules. Each rule is defined in terms of the static IP address bound to the virtual interface assigned to a Mon-VM. Every packet with a spoofed source IP address is dropped and logged.

4 Security and Performance Results

This section presents an evaluation of Psyco-Virt from the security and the performance points of view. To test Psyco-Virt capabilities, we configured an IVM to compute, with a predefined frequency, the hashes of the text area of the kernel and of the modules inside each Mon-VM. In turn, an authorized module into the kernel of each Mon-VM periodically computes the hashes of the text area of critical processes of a Mon-VM, such as the collector, the hosts and network IDSes. The IVM also verifies that the list of the loaded modules into the kernel only contains authorized modules, and that the list of running processes and open files does not include hidden entries. Lastly, it applies anti-spoofing techniques on the virtual bridge and checks that no Mon-VM is sniffing traffic.

4.1 Effectiveness

To evaluate the effectiveness of the checks implemented by Psyco-Virt, we wrote sample rootkits to update both the text area of a critical process and an entry in the *idt_table* pointing to a modified interrupt handler [17]. We also inserted a malicious module into the kernel of each Mon-VM to modify both a pointer in the *sys_call_table* and an existing system call. Besides, we replaced the system binaries *ps* and *lsof* to hide specific processes/files.

The IVM detects the modifications to the system call handler or to an existing system call as well as to pointers in the *sys_call_table* or to an entry in the *idt_table*. The module into the Mon-VM kernel also detects when the text area of a critical process is modified, while the IVM detects each not authorized module loaded into the Mon-VM kernel, as well as a malicious update to an authorized module. Moreover, chkrootkit or Tripwire detects any update to system binaries and the process checker discovers the presence of hidden processes/files. In any of the previous cases, an alert is sent to the director. Finally, the IVM easily detects when an interface is set to promiscuous mode, or when the IP source address of a packet has been spoofed.

4.2 Performance Overhead

To evaluate the overhead due to Psyco-Virt, we ran the IOzone Filesystem benchmark tool [14] on a Mon-VM, while the IVM applies the whole set of consistency checks discussed earlier, with a period between each invocation of 1, 5, 10, 30, 60 seconds or without applying any check. Figure 2 shows that the maximum overhead on the read test, due to the consistency checks, is about 7%. The same result holds for the write test.

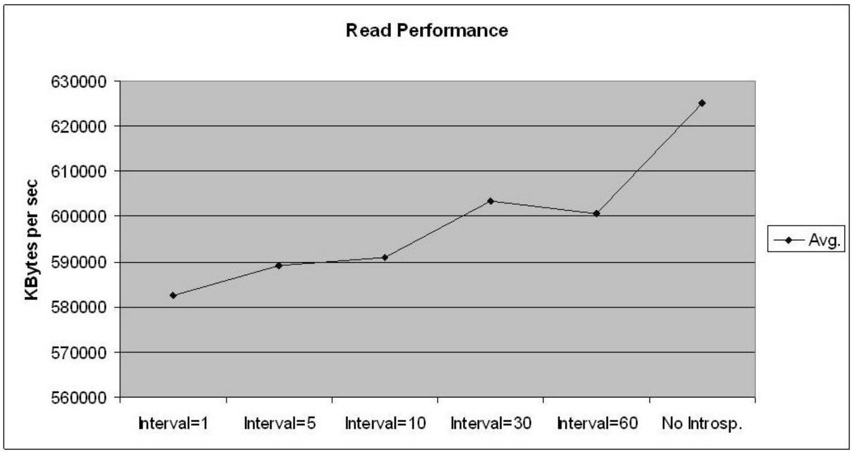


Fig. 2. IOzone read performance on a Mon-VM.

4.3 Limitations

Currently there are some attacks that Psyco-Virt cannot detect. As an example, Psyco-Virt only checks that the *idt_table* and the *sys_call_table* have not been updated, but several other kernel sensitive data structures need to be protected. Moreover, any malicious modification against dynamic data in memory, both in the kernel and user space, is not detected. As an example, malicious updates of the stack or of the heap are not detected. The complexity of preventing these attacks is very high because a set of memory invariants has to be computed for each process and for the kernel. Another problem arises if an attacker detects the presence of the VMM. In fact, she can try to directly attack the VMM or evade the consistency checks. Evasion is possible provided that sensitive data structures are updated after the checks have been applied and that a consistent state is restored just before the checks are applied again. Finally, provided that an attacker can insert a module the first time the hashes of the authorized modules are computed, a malicious module can be considered an authentic one.

5 Related Works

VMI is first discussed in [12] together with *Livewire*, which is a prototype of VMI IDS. *ReVirt* [8] supports *recovery*, *checkpoint* and *roll-back* of VMs and uses *virtual-machine replay* to re-execute a system, encapsulated in a VM, instruction-by-instruction for recovering purposes. *IntroVirt* [16] is a system that detects intrusions by executing vulnerability-specific predicates. *Paladin* [1] is a framework that exploits the virtualization technology to detect and contain rootkit attacks. *Manitou* [19] is a system implemented within a VMM that ensures that a VM can only execute authorized code by computing the hash of each page before executing the code and setting the executable bit only if its hash belongs to a list of authorized hashes. [35] describes a hierarchical trust management framework, where the root of trust is a secure co-processor, which periodically triggers a set of security checkers to build up a chain of trust. The idea of a distributed IDS was first introduced in [29] to monitor a heterogeneous network. The proposed prototype combined data reduction and a centralized data analysis. *Netstat* [32] is a NIDS that applies state transition analysis techniques by modeling intrusions through state transition diagrams. *Hyperspector* [18] is a monitoring environment to detect intrusions in a distributed system, by running each IDS inside a dedicated VM, and connecting all the IDS using an independent virtual network. *Collapsar* [15] is a virtual honeypot architecture to detect network attacks.

6 Conclusions and Future Developments

Psyco-Virt shows that the proposed multi-level approach to achieve highly assurance intrusion detection systems is feasible and effective, and that the overhead is acceptable. The proposed approach defines a three-steps strategy: (i) a set of processes implements HIDS and NIDS tools to detect attacks and intrusions on the Mon-VMs; (ii) specific modules into the kernel of the Mon-VMs check the integrity of these critical processes; (iii) virtual machine introspection is used to protect kernel integrity. In this way, the proposed architecture builds a chain of trust, where each level verifies the integrity of the one above it.

Our future research is focused on the use of introspection to check at runtime a set of memory invariants that should hold for a process or for the kernel. These invariants are computed by applying formal static analysis based on an abstract interpretation approach to the programs or to the kernel code [6, 3, 2]. The set of invariants is an input for the IVM which, for example, may freeze a Mon-VM each time a system call is executed, and apply introspection to check that invariant properties regarding system call parameters are verified.

Acknowledgments

We would like to thank Fabio Campisi which helped us in writing and testing the code and the anonymous reviewers for their suggestions.

References

[1] A Baliga, X Chen, and L Iftode. Paladin: Automated detection and containment of rootkit attacks, Jan 2006. Rutgers University Department of Computer Science Technical Report DCS-TR-593.

[2] T Ball, R Majumdar, T D Millstein, and S K Rajamani. Automatic predicate abstraction of c programs. In SIGPLAN Conference on Programming Language Design and Implementation, pp. 203–213, 2001.

[3] F Bourdoncle. Abstract debugging of higher-order imperative languages. In PLDI ’93: Proceedings of the ACM SIGPLAN 1993 conference on Programming language design and implementation, pp. 46–55, New York, NY, USA, 1993. ACM Press.

[4] R Bradshaw, N Desai, T Freeman, and K Keahey. A scalable approach to deploying and managing appliances. In TeraGrid 2007, June 2007.

[5] chkrootkit – locally checks for signs of a rootkit. http://www.chkrootkit.org/.

[6] P Cousot and R Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In POPL, pp. 238–252, 1977.
[7] B Dragovic, K Fraser, S Hand, T Harris, A Ho, I Pratt, A Warfield, P Barham, and R Neugebauer. Xen and the art of virtualization. In Proceedings of the ACM Symposium on Operating Systems Principles, October 2003.
[8] G W Dunlap, S T King, S Cinar, M A Basrai, and P M. Chen. ReVirt: enabling intrusion analysis through virtual-machine logging and replay. In Proceedings of the 5th Symposium on Operating Systems Design and Implementation, pp. 211–224, New York, NY, USA, 2002. ACM Press.
[9] T Fraser. Automatic discovery of integrity constraints in binary kernel modules, Technical report, University of Maryland Institute for Advanced Computer Studies, December 2004
[10] The FU rootkit. http://www.rootkit.com/project.php?id=12.
[11] FuSyS. Kstat. http://www.s0ftpj.org/tools/kstat24 v1.1-2.tgz.
[12] T Garfinkel and M Rosenblum. A virtual machine introspection based architecture for intrusion detection. In Proceedings of the Network and Distributed Systems Security Symposium, February 2003.
[13] R P Goldberg. Survey of virtual machine research. IEEE Computer, 7(6):34–45, 1974.
[14] IOzone Filesystem Benchmark, http://www.iozone.org/.
[15] X Jiang and D Xu. Collapsar: A VM-based architecture for network attack detention center. In USENIX Security Symposium, pp. 15–28, 2004.
[16] A Joshi, S T King, G W Dunlap, and P M Chen. Detecting past and present intrusions through vulnerability specific predicates. In SOSP '05: Proceedings of the Twentieth ACM symposium on Operating systems principles, pp. 91–104, New York, NY, USA, 2005. ACM Press.
[17] kad. Handling Interrupt Descriptor Table for fun and profit. Phrack, 11(59), July 2002.
[18] K Kourai and S Chiba. HyperSpector: virtual distributed monitoring environments for secure intrusion detection. In VEE '05: Proceedings of the 1st ACM/USENIX international conference on Virtual execution environments, pp. 197–207, New York, USA, 2005. ACM Press.
[19] L Litty and D Lie. Manitou: a layer below approach to fighting malware. In ASID '06: Proceedings of the 1st Workshop on Architectural and System Support for Improving Software Dependability, pp. 6–11, New York, USA, 2006. ACM Press.
[20] P Loscocco and S Smalley. Integrating flexible support for security policies into the linux operating system. In Proceedings of the FREENIX Track: 2001 USENIX Annual Technical Conference, pp. 29–42, Berkeley, CA, USA, 2001. USENIX Association.
[21] P A Loscocco and S D Smalley. Meeting critical security objectives with security enhanced linux. In Proceedings of the 2001 Ottawa Linux Symposium, 2001.
[22] Netfilter/Iptables project. www.netfilter.org/.
[23] Open source tripwire. http://sourceforge.net/ projects/tripwire/.

[24] Openssl: The open source toolkit for ssl/tls. http://www.openssl.org/.
[25] OpenVPN - An Open Source SSL VPN Solution. http://openvpn.net/.
[26] N L Petroni, T Fraser, J Molina, and W A Arbaugh. Copilot - a coprocessor-based kernel runtime integrity monitor. In USENIX Security Symposium, pp.179–194, 2004.
[27] C Sapuntzakis, D Brumley, R Chandra, N Zeldovich, J Chow, M Lam, and M Rosenblum. Virtual appliances for deploying and maintaining software, 2003.
[28] sd and devik. Linux on-the-fly kernel patching without LKM. Phrack, 10(58), December 2001.
[29] S R Snapp, J Brentano, G V Dias, T L Goan, L T Heberlein, C Ho, K N Levitt, B Mukherjee, S E Smaha, T Grance, D M Teal, and D Mansur. DIDS (Distributed Intrusion Detection System) - motivation, architecture, and an early prototype. In Proceedings of the 14th National Computer Security Conference, pp. 167–176,Washington, DC, 1991.
[30] Snort - the de facto standard for intrusion detection/prevention. http://www.snort.org/.
[31] S Sparks and J Butler. Shadow Walker: Raising the bar for rootkit detection. www.blackhat.com/presentations/bh-jp-05/bh-jp-05-sparks-butler.pdf.
[32] G Vigna and R A. Kemmerer. Netstat: A network-based intrusion detection system. Journal of Computer Security, 7(1), 1999.
[33] VMTN - Virtual Appliance Marketplace. http://www.vmware.com/vmtn/appliances/.
[34] VMware. http://www.vmware.com/.
[35] L Wang and P Dasgupta. Kernel and application integrity assurance: Ensuring freedom from rootkits and malware in a computer system. In AINAW '07: Proceedings of the 21st International Conference on Advanced Information Networking and ApplicationsWorkshops, pp. 583–589,Washington, DC, USA, 2007. IEEE Computer Society.
[36] XenAccess Library. http://xenaccess.sourceforge.net/.

3
Intrusion Detection Using Cost-Sensitive Classification

Aikaterini Mitrokotsa[1,*], Christos Dimitrakakis[2], and Christos Douligeris[3]

[1] Vrije University Amsterdam, Netherlands mitrokat@unipi.gr
[2] University of Leoben, Leoben, Austria, christos.dimitrakakis@mu-leoben.at
[3] University of Piraeus, Piraeus, Greece, cdoulig@unipi.gr

Abstract. Intrusion Detection is an invaluable part of computer networks defense. An important consideration is the fact that raising false alarms carries a significantly lower cost than not detecting attacks. For this reason, we examine how cost-sensitive classification methods can be used in Intrusion Detection systems. The performance of the approach is evaluated under different experimental conditions, cost matrices and different classification models, in terms of expected cost, as well as detection and false alarm rates. We find that even under unfavourable conditions, cost-sensitive classification can improve performance significantly, if only slightly.

1 Introduction

Standard classification problems require making the classification decision that minimizes the probability of error. However, for many problem domains, the requirement is not merely to predict the most probable class label, since different types of errors carry different costs. Instances of such problems include authentication, where the cost of allowing unauthorized access can be much greater than that of wrongly denying access to authorized individuals, and intrusion detection, where raising false alarms has a substantially lower cost than allowing an undetected intrusion. In such cases, it is preferable to make the classification decision that has minimum expected cost, rather than that with the lowest error probability.

While there has been an extensive body of work in this field, particularly in the domain of optimal statistical decisions (see for example [1] for

[*] Work done while Aikaterini Mitrokotsa was with the University of Piraeus.

an overview), this has been largely ignored in the domain of intrusion detection. We are currently aware of two other papers ([2], [3]) dealing with cost-sensitive intrusion detection, both using a wrapper algorithm (MetaCost [4] and Weighted [5] respectively) together with RIPPER [6]. Although both papers report results on the KDD database, neither does so for the given cost matrix, so direct comparisons between the statistical models employed herein and the wrapper algorithms are not possible. This paper attempts to answer some very basic questions about cost-sensitive classification. Firstly, to what extent must the test distribution match the training data distribution. Secondly, for the dataset used, are some methods consistently better than others, or is there some variability and why. Finally, to what extent does the false alarm rate grow when the cost of missed attacks rises with respect to the cost of false alarms.

The next section discusses how to use classification methods that can be readily embedded in the formal optimal statistical decision framework in order to create intrusion detection systems that will be effective in minimizing the expected cost of their operation and analyses the relationship between cost matrices and the desired trade-off between detection and false alarm rates. Finally, it gives a brief introduction to the classification models used. Section 3 outlines the experiments performed and we conclude with a discussion on the significance of the results and on future research directions.

2 Cost Sensitive Classification

Given a specification of costs for correct and incorrect predictions, the class decision should be the one that leads to the lowest expected cost, where the expectation is computed using the conditional probability of each class given the example, according to our model[1]. More formally, for a set Ω of k classes let a k x k matrix C such that $C(i, j)$ is the expected cost of predicting class i when the true class is j. If $i = j$ then the decision is correct, while if $i \neq j$ the decision is incorrect. Furthermore, let Y, H be random variables denoting the actual and hypothesized class labels. For any observations $x \in X$ the optimal decision will be the class i that minimizes a loss function equal to the expected cost

$$L(x,i) = E[C \mid X = x, H = i] \equiv \sum_{j \in \Omega} P(Y = j \mid X = x) \cdot C(i.j) \tag{1}$$

[1] The implicit dependency on some model m can be made explicit by conditioning everything on the model. Then the expectation would be written $E[C \mid x, f, m]$ and the conditional class probability $P(y \mid x, m)$.

where $P(Y \mid X)$ denotes the conditional distribution of class labels given an observation, according to our model. In this framework, all that is necessary is a model that can estimate this probability. The cost-sensitive decision-making function $f{:}S \rightarrow \Omega$ would then simply chose the decision i that minimises the expected cost given the decision and the example[2]. More formally,

$$f(x) = \underset{i \in \Omega}{arg\,min}\, L(x,i) \tag{2}$$

The form of the cost matrix C will depend on the actual application. In general, it is reasonable to choose the diagonal entries equal to zero, i.e. $C(i, j) = 0$ for $i = j$, since correct classification normally incurs no cost. The other entries specify the cost of incorrectly misclassifying an example of class j as belonging to class i. They should be non-negative if the diagonal is zero, i.e. $C(i, j) \geq 0$ for $i \neq j$. Note that when this is equal to 1, the cost measure is the same as the classification error measure.

2.1 Choice of the Cost Matrix

As an example, consider a cost matrix C for two classes, positive and negative. The cost of a false positive is $C(2, 1)$, while that of a false negative is $C(1, 2)$ and we can set $C(1, 1) = C(2, 2) = 0$, i.e. a correct classification will have no cost. For intrusion detection applications, it is common to refer to attacks as positive and normal instances as negative example. Furthermore, the occurrence of false negatives (FN) is usually considered a worse kind of error than that of a false positive (FP), thus the matrix C should reflect that, by having $C(1, 2) \geq C(2, 1)$. In some cases, such as in some benchmark databases for intrusion detection, the cost matrix is given, while in others it must be chosen by the user.

2.2 Algorithmic Comparisons and Alternative Quality Metrics

When comparisons are made between algorithms, it is important to use the same measure of quality for all of them. A common measure of quality is the empirical value of the expected value of the cost C measured over an independent test set D,

$$\hat{E}(C \mid D) = \frac{1}{|D|} \sum_{d \in D} C(f(x_d), y_d), \tag{3}$$

[2] Which of course is not necessarily identical to the decision with the minimum error probability. Furthermore, this framework is easily extensible to the case where the set of decisions differs from the set of class labels.

where $d \equiv (x_d, y_d)$. Whenever such a cost matrix is set as the evaluation metric in a benchmark database, then it is preferable to use it. However, it is important to note that in much of the literature, the following pair of measures is used instead. The *Detection Rate (DR)*, and the *False Alarm* (FA) rate

$$DR = \frac{TP}{TP + TN}, \quad FA = \frac{FP}{TN + FP} \tag{4}$$

where *TP, TN, FP, FN,* denote the number of true *(T)* and false *(F)* positives and negatives respectively. The aim would be to reduce *FA* rate, while at the same time increasing *DR*. Since this is not usually possible, a trade-off between the two quantities is often sought instead. While such a trade-off may be automatically accomplished through the use of an appropriate cost matrix[3], in this paper we will only use these quantities as a secondary alternative comparison metric.

2.3 Models

As mentioned in the beginning of Section 2, the computation of class probabilities is model-dependent. Ideally one would assume a Bayesian viewpoint and consider a distribution over all possible models in a set of models, however in this case we will only consider point distributions in model space, i.e. a single parameter vector in parameter space. While this can cause problems with overfitting, we will use frequentist model selection methods to avoid this potential pitfall. These are described further in Section 3.2. The rest of this section gives a brief overview of the two models used in this work, the multilayer perceptron (MLP) and the Gaussian mixture model (GMM). A specific instance of an MLP can be viewed simply as a function $g{:}S \to \Omega$, where g can be further defined as a composition of other functions $z_i{:}S \to Z$. In most cases of interest, this decomposition can be written as $g(x) = K(w'z(x))$, with $x \in S$, w being a parameter vector, while K is a particular kernel and the function $z(x) = [z_1(x), z_2(x), \ldots]$ is referred to as the *hidden layer*. For each of those, we have $z_i(x) = K_i(v_i' x)$, where each v_i is a parameter vector, $V = [v_1, v_2, \ldots]$ is the parameter matrix of the hidden layer and finally K_i is

[3] Let the expected cost be E[C] = qP(H = 1|C = 2)P(C = 2) +r P(H = 2|C = 1)P(C =1) = q(1 – DR)P(C = 2) + rFA P(C = 1), where 2 denotes a positive example. Setting r = 1/P(C = 1) and q = k/P(C = 2) we obtain a cost function minimizing FA – kDR, with *k* being a free parameter specifying the trade-off we are interested in.

an arbitrary kernel. For this particular application wish to use an MLP *m* as a model for the conditional class probability given the observations, i.e.

$$P(Y = y \mid X = x, M = m), \qquad y = g(x), \tag{5}$$

for which reason we are using a sigmoid kernel for *K*. In the experiments we shall be employing a hyperbolic tangent as the kernel for the hidden layer, when there is one. The case where there is no hidden layer is equivalent to $z_i = x_i$ and corresponds to the *linear* model. The GMM, the second model under consideration, will be used to model the conditional observation density for each class, i.e.

$$P(X = x \mid Y = y, M = m) \tag{6}$$

This can be achieved simply by using a separate set of mixtures U_y for modeling the observation density of each class *y*. Then, for a given class y the density at each point *x* is calculated by marginalizing over the mixture components $u \in U_y$, for the class, dropping the dependency on *m* for simplicity.

$$P(X = x \mid Y = y) = \sum_u P(X = x \mid U = u) P(U = u \mid Y = y) \tag{7}$$

Note that the likelihood function $p(X = x \mid U = u)$ will have Gaussian form, with parameters Σ_u, the covariance matrix and μ_u the mean vector, while $P(U = u \mid Y = y)$ will be another parameter, the component weight[4]. Finally, we must separately estimate *P(Y = y)* from the data, thus obtaining the conditional class probability given the observations

$$P(Y = y \mid X = x) = \frac{1}{Z} P(X = x \mid Y = y) P(Y = y) \tag{8}$$

where $Z = \sum_{j \in \Omega} P(X = x \mid Y = y) P(Y = j)$ does not depend on *y* and where we have again dropped the dependency on *m*.

The conditional class probabilities from either (7) or (8), depending on the model, can then be plugged into (1), for calculating the decision function (2).

3 Experiments

In order to examine the effectiveness of the proposed approach, we conducted a series of experiments under varying conditions. In our experi-

[4] Since we use separate mixture components for each class, $P(U = u \mid Y = y) = 0$, when $u \notin U_y$, which also allows us to drop the dependency on *y* in the likelihood function.

ments we performed comparisons in terms of the weighted cost defined in equation (3) using four different models: the MLP, Linear, GMM with diagonal covariance matrixes and Naïve Bayes models (GMM with a single Gaussian). It was expected that using the cost matrix to make decisions would result in a lower cost than when not doing so, even if the models' class probability estimates are not very accurate. A particularly interesting question was how the divergence between the training and testing data distributions affects the measured cost, for a fixed cost matrix. We furthermore investigated how the false alarm and detection rates change when we vary the relative cost of false alarms and false negatives.

3.1 Databases

We performed our experiments on the KDD database [7], using the 10% KDD dataset for training and cross-validation. The KDD dataset include four types of attacks Denial of Service (DoS), Remote to Local (R2L), User to Root (U2R) and Probe.

Denial of Service (DoS): The main aim of a DoS attack is the disruption of services by attempting to limit access to a machine or service. Examples are back, land pod teardrop, smurf and neptnune.
Remote to Local (R2L): In a remote to local attack the attacker gains unauthorized local access from a remote machine and exploits this access in order to send packets over the network. Examples are Ftp_write, Guess passwd, Imap, warezclient, warezmaster, phf, spy and multihop.
User to Root (U2R): In U2R the attacker gains unauthorized access to local super user (root) privileges. Examples are Loadmodule, Perl, rookit and buffer overflow.
Probe: the attacker scans a network in order to find vulnerabilities requires little technical expertise. Examples are ipsweep, nmap, portsweep and satan.

Furthermore we performed two evaluations. For the first evaluation, we used the standard test KDD dataset, which includes 311,029 connections, including 17 types of attacks which are never observed in the training dataset. More specifically, in the full test data there are 4 new U2R attacks that correspond to the 92.90% (189/228) of the U2R class, 7 new R2L attacks that correspond to 63% (10196/16189) of R2L class in that dataset, 4 new DoS attacks that correspond to 2.85% (6555/229853) of the DoS class and 2 new types of Probe attacks that correspond to 42.94% (1789/4166) of all the Probe attacks. For this reason, for our second evaluation we used a version of this dataset which does not include these novel attacks. In both cases, the probability distribution of the test datasets is not the same as that of the training dataset.

The datasets are summarised in Table 1.

Table 1. Proportion of attack and normal connections for training and testing datasets.

Datasets	Probe	DoS	R2L	U2R	Total attacks	Total normal
10% KDD Dataset	4107	391458	1126	52	396743	97278
Test Dataset 1	4166	229853	16189	228	250436	60593
Test Dataset 2	2377	223298	5993	39	231707	60593

3.2 Technical Details

In order to select the best parameters for each model we performed 10-fold cross validation. For each MLP model we tuned three parameters, the *learning rate (η)*, the number of *iterations (T)* and the number of *hidden units (n_h)*. Keeping stable the *n_h* (equal to 0) we selected the appropriate *η* among values that range between 0.0001 and 0.1 with step 0.1 and the appropriate *T* selecting among 10, 100, 500 and 1000. For the selection of the appropriate *n_h*, having selected the appropriate *η* and the appropriate of *T*, we examined various values for *n_h* and selected the best among 10, 20, 40, 60, 80, 100, 120, 140, 160, 320. We additionally, used the MLP model with no *hidden units* as a *Linear* model.

For the GMM model we also tuned three parameters the *threshold (θ)*, the number of *iterations (T)* and the number of *Gaussian Mixtures (n_g)*. Keeping stable the *n_g* (equal to 20) we selected the appropriate *θ* among values that range between 0.0001 and 0.1 with step 0.1 and the appropriate *T* among 25, 100, 500 and 1000. For the selection of the appropriate *n_g*, after selecting the appropriate *θ* and the appropriate *T*, we examined various values for the *n_g* and the selected the best among 10, 20, 40, 60, 80, 100, 120, 140, 160, 320. We additionally, used the GMM model with one Mixture component as a Naïve Bayes model.

We have used the cost matrix defined for the KDD 1999 Dataset [8], which is shown in Table 2. We have also defined an arbitrary table in order to examine how the measured cost changes when the relative cost for the misclassification of attack versus normal connection increases. Thus, we define a 5x5 cost matrix *A* where *A(j,1)=α*, and *A(1,i)=1* for *j=2,3,4,5* and *i=2,3,4,5*. Also *A(i,j)=0*, for *i=2,3,4,5* and *j=2,3,4,5*, *A(1,1)=0* and *a* take values between 1 and 10 with step 1. Since the fields of the KDD dataset include discrete and continuous values, we represent the discrete values using one hot encoding. Furthermore, to ensure a good behaviour of all training algorithms we have normalized all the datasets to zero mean and unit variance.

Table 2. Cost matrix for the KDD 99 dataset.

Actual \ Predicted	Normal	Probe	DoS	U2R	R2L
Normal	0	1	2	2	2
Probe	1	0	2	2	2
DoS	2	1	0	2	2
U2R	3	2	2	0	2
R2L	4	2	2	2	0

3.3 Results

We have evaluated each algorithm both with and without the use of a cost matrix for making decisions. Table 3 shows the results for Dataset 1, while Table 4 for Dataset 2. In both cases, μ is the expected cost, while low and high are the boundary values of the 99% confidence interval. The latter was estimated using 1000 bootstrap [9] samples of the test datasets.

It is clear that the empirical average cost for Dataset 1 is much higher than the corresponding cost for Dataset 2. This was expected, since Dataset 1 includes types of attacks that were not included in the training data. It is also evident that the Linear and GMM classifiers both achieve better results when we are using the cost matrix to make decisions. However, this was not the case for either the MLP or the Naive Bayes classifier. The latter's failure could be attributed to the fact that the Naive classifier assumes

Table 3. Expected cost (μ) and boundary values (Low, High) with confidence 99% for Testing Dataset 1 with and without the use of a cost matrix.

	Without cost			With cost		
	Low	μ	High	Low	μ	High
MLP	0.2384	0.2427	0.2472	0.2390	0.2431	0.2476
Linear	0.2425	0.2467	0.2511	0.2414	0.2452	0.2489
GMM	0.2497	0.2538	0.2578	0.2378	0.2420	0.2457
Naïve Bayes	0.3786	0.3829	0.3871	0.5304	0.5400	0.5353

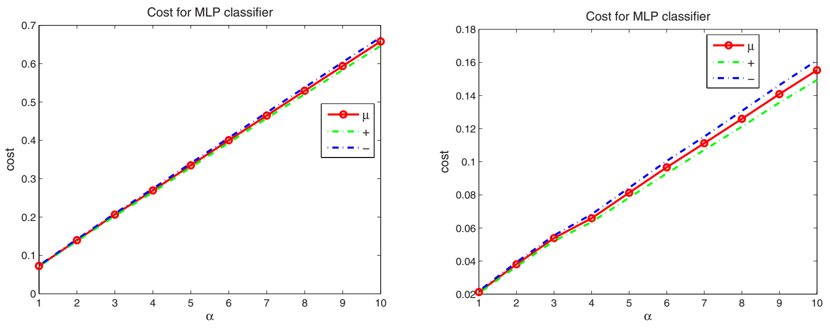
Table 4. Expected cost (μ) and boundary values (Low, High) with confidence 99% for Testing Dataset 2 with and without the use of a cost matrix.

	Without cost			With cost		
	Low	μ	High	Low	μ	High
MLP	0.0711	0.0736	0.0759	0.0713	0.0739	0.0765
Linear	0.0735	0.0761	0.0784	0.0716	0.074	0.0763
GMM	0.0821	0.0845	0.0869	0.0686	0.071	0.0734
Naïve Bayes	0.2173	0.2204	0.2231	0.3733	0.3775	0.3817

that all features of the training Dataset are independent. The reason for the behaviour of the MLP is not entirely clear. One possibility is that the probabilities that the model outputs do not accurately represent the uncertainty of classification, i.e. the classifier is ‘too confident’ due to the maximum likelihood training. However, we also note that nevertheless the MLP model performed as well as the weighted GMM model. This hypothesis is consistent with the fact that the MLP nevertheless performed as well as the weighted GMM model in Dataset 1, but significantly worse in Dataset 2.

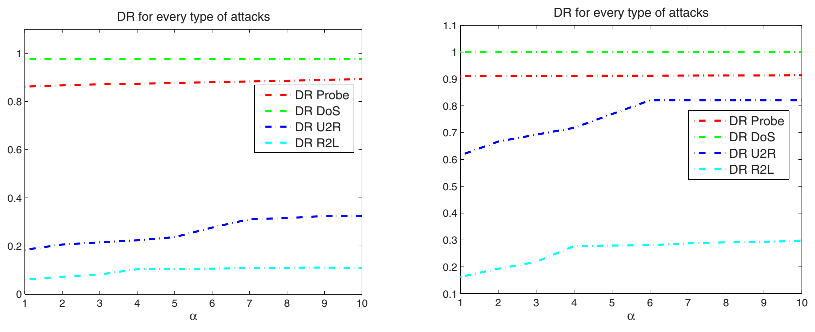
We have performed a series of experiments for the MLP classifier, since this one presents the lower classification error, for the arbitrary cost matrix described in Section 3.2. We examined how the performance of the MLP changes when we increase the cost (α) of false positives relative to that of false negatives. Again, we used the boostrap methodology to obtain confidence intervals for the results.

In Fig. 1 (a) and (b) the middle line (μ) represents the expected cost as it was estimated for the testing Datasets 1 and 2 respectively. The surrounding lines denote the 99% confidence interval of the expected cost as this was estimated from the bootstrap samples. From Fig. 1 (a) and (b), it is clear that the expected cost for Datasets 1 and 2 respectively, increases linearly with α, which indicates a good behavior of the classifier.



(a) Expected Cost for Testing Dataset 1 (b) Expected Cost for Testing Dataset 2

Fig. 1. Expected cost (μ) for Testing Datasets 1 **(a)** and 2 **(b)** respectively, with the cost of false negatives (attacks detected as normal) being equal to α, and the cost of false positives (false alarms) set equal to 1. The dashed lines (+,–) represent the bootstrap estimate of the 99% confidence intervals.



(a) Detection Rate for Testing Dataset 1 (b) Detection Rate for Testing Dataset 2

Fig. 2. The Detection Rate (DR) evaluated on Testing Datasets 1 **(a)** and 2 **(b)** respectively, with the cost of false negatives (attacks detected as normal) being equal to α, and the cost of false positives (false alarms) set equal to 1.

In Fig. 2 (a) and (b) we observe the detection rate for each type of attack for both testing datasets. The Detection Rate (DR) of all attacks is better for Dataset 2 while there is an increase of the Detection Rate for all type of attacks and both Datasets. While we observe a significant increase in the detection rate for the attacks in the training Dataset, this is not the case for the novel attacks, especially for the R2L attacks. The Detection Rate for DoS and Probe attacks presents only a slight increase for both cases. While for U2R attacks and Dataset 1 the Detection Rate ranges from 0.184 to 0.325 an increase of 14.1%. For U2R attacks and Dataset 2 we also observe a substantial increase of the Detection Rate from 0.615 to 0.82, an increase of 20%. For R2L attacks and Dataset 1 there is an increase of the Detection Rate from 0.06 to 0.108, an increase of 10.2%. For R2L attacks and Dataset 2 the Detection Rate ranges from 0.163 to 0.297, an increase of 13.4%.

In Fig. 3 we observe that the average Detection Rate (DR) presents a slight increase from 0.913 to 0.919 for Dataset 1 and from 0.977 to 0.98 for Dataset 2. Figure 4 depicts how the False Alarm (FA) rate is influenced by the increase of the cost of false positives relative to that of false negatives. We observe a slight increase from 0.016 to 0.018, thus the increase is of 0.2%. The False Alarm (FA) rate is the same for both Datasets since the number of normal connections is the same in both Datasets. Overall, the detection was increased significantly for the U2R (20%) and R2L (13.4%) attacks for Dataset 2, but *not* for novel attacks in these categories. In any case, the increase in false alarms was only 0.2%.

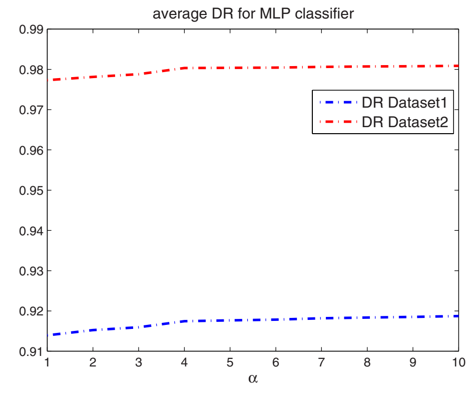


Fig. 3. The average DR for each Testing Dataset using the MLP classifier for various values of the cost of false positives relative to that of false negatives (α).

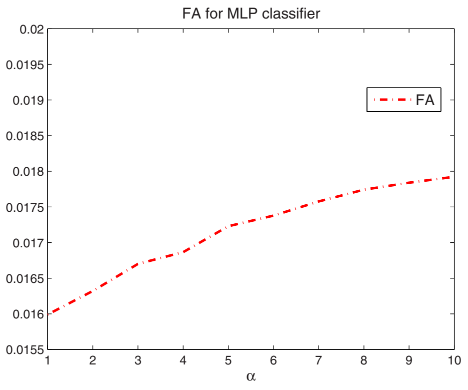


Fig. 4. The FA rate for each Testing Dataset using the MLP classifier for various values of the cost of false positives relative to that of false negatives (α).

4 Conclusions

The experimental results indicate that cost-sensitive classification methods using standard statistical classifiers to estimate class probabilities can behave quite well even in some cases where assumptions about the test data distribution are violated. This is not true for all methods tested. For the Naïve model, this can be explained by the fact that the assumption of feature independence cannot be maintained. However the behaviour of the MLP model is not as easy to explain. One possible explanation could have been that the predicted class probabilities by the MLP are more close to 1 than is warranted. However, Figs. 1–4 do not give support to this hypothesis. On the other hand, while the GMM's cost is significantly reduced when using weighted decisions, as can be seen by the lack of overlap between the confidence intervals, the MLP (whether weighted or not) is performing just as well as the weighted GMM in terms of cost. A possible explanation then is that the training and test data distributions are different enough for the generalization ability of a classifier to be more important than the use of the correct cost matrix.

Future work would have to further examine the relationship between distribution divergence and the use of cost matrices. An interesting approach would be to use fully Bayesian methods, and also to perform more complete comparisons. A final point that we have only touched upon in the introduction is that the set of actions does not necessarily have to coincide with the set of classes. Then we would make decisions that minimise the expected cost of each decision. Examples of such decisions would be "Do nothing", "Call Administrator", "Block IP Address". Furthermore, we could even consider intrusion detection as a sequential decision making [1] problem, where each decision would not only depend upon the current observation, but on the history of observations and past decisions. This would not only make such systems more flexible, but could also reduce much of the future engineering performed in order to select what is the best time window in which to collect packet statistics.

References

[1] DeGroot MH (2004) Optimal Statistical Decisions. John Wiley & Sons, New York. 1970. Republished in 2004

[2] Fan W, Lee W, Stolfo SJ, Miller M (2000) A multiple model cost-sensitive approach for intrusion detection. In: Proceedings of the 11th European conference on Machine Learning 2000 (ECML'00), Barcelona, Catalonia, Spain, Lecture Notes in Computer Science, vol. 1810, pp 142–153

[3] Pietraszek P (2004) Using adaptive alert classification to reduce false positives in intrusion Detection. In: Proceedings of Recent Advances in Intrusion Detection 7th International Symposium (RAID'04), Sophia, Antipolis, France, Lecture Notes in Computer Science 3224, Springer, pp102–124

[4] Domingos P (1999) MetaCost A general method for making classifiers cost-sensitive. In: Proceedings of the Fifth ACM SIGKDD Int'l conf. On Knowledge Discovery and Data Mining, San Diego, CA, pp 155–164

[5] Ting K (1998) Inducing cost-sensitive trees via instance weighting. In: Proceedings of the Second European Symposium on Principles of Data Mining and Knowledge Discovery. vol 1510 of Lecture Notes in AI., Springer-Verlag, pp 137–147

[6] Cohen WW (1995) Fast effective rule induction. In: Proceedings of the Twelfth International Conference on Machine Learning, Lake Taho, CA, Morgan Kaufmann, pp 115–123
[7] KDD Cup 1999 Data (1999). Available from <http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html>
[8] Elkan C (1999) Results of the KDD'99 Classifier Learning Contest. September, Available from < http://www-cse.ucsd.edu/users/elkan/clresults.html>
[9] Efron B, Tibshirani RJ (1994) An Introduction to the Bootstrap. Monographs on Statistics & Applied Probability, vol. 57, Chapmann & Hall, New York, Nov, Pub.

4
A Novel Approach for Anomaly Detection over High-Speed Networks

Osman Salem, Sandrine Vaton, and Annie Gravey

ENST Bretagne, Department of Computer Science, Brest France
{osman.salem, sandrine.vaton, annie.gravey}@enst-bretagne.fr

Abstract. This paper provides a new framework for efficient detection and identification of network anomalies over high speed links, in early stage of its occurrence to quickly react by taking the appropriate countermeasures. The proposed framework is based on change point detection in counters value of reversible sketch, which aggregates multiple data streams from high speed links in a stretched database. To detect network anomalies, we apply the cumulative sum (CUSUM) algorithm at the counter value of each bucket in the proposed reversible sketch, to detect change point occurrence and to uncover culprit flows via a new approach for sketch inversion. Theoretical framework for attacks detection is presented. We also give the results of our experiments analysis over two real data traces containing anomalies, and extensively analyzed in OSCAR French research project. Our analysis results from real-time internet traffic and online implementation over Endace DAG 3.6ET card show that our proposed architecture is able to detect culprit flows quickly with a high level of accuracy.

1 Introduction

Security threats for computer network have increased significantly, which include viruses, worm-based attacks, PortScan, NetScan, denial of service (DoS), and its distributed version (DDoS), etc. However, with the increasing of link speeds and traffic volume, the real time monitoring and analyzing of IP traffic to detect attacks become a complicated task, but crucial for managing large networks (e.g. for ISP, Enterprise, etc.).

Two approaches to network anomaly detection are used. The first is signature based-approach, which is extensively explored in many software systems and toolkits such as Bro [17] and Snort [18]. This approach is used for anomaly detection with signatures known in advance, and it can not be applied to identify new anomalies.

The second approach is the statistics based approach, which does not require prior knowledge about the nature and properties of anomalies and therefore can be effective even for new anomalies or variants of existing anomalies. A very important component of statistics based approach is change detection [11]. It builds a model for normal user behavior in learning phase, and any inconsistent behavior with the build model is considered as anomaly. While wide range anomaly detection algorithms have been introduced to undermine attacks, the effectiveness of these models is largely dependent of traffic distributions parameters and their variations. They lack the capability of handling shape irregularities and unpredictable large fluctuations of real IP traffic.

Furthermore, most existing intrusion detection systems (IDS) reside at end host or end router. They lack scalability in handling large state space traffic information at high speed links, where even the handling at flow level is very costly in term of per-flow storage requirement, and update/search operations complexity. Flows are usually characterized by 5 fields (e.g. Netflow [2]): source and destination IP address, source and destination port, and protocol number. This means monitoring flows state space requires updating and handling a database table of size 2^{104}.

A naïve idea for monitoring flows over high speed links is to maintain a database for active flows in a fixed time interval *T*, and to track the *k* frequent destination addresses in IP traffic (top ten or heavy hitter destinations). For example, to detect victim servers of SYN flooding DDoS attack, an ISP can query database for destinations IP with a received number of SYN that exceeds a given percentage of the total number of SYN, which was relayed by the monitoring node. However, this strategy is not scalable, where spatial and temporal complexities for update and query operations, on active flows database prevent its use for handling a large number of flows at high speed links in real time.

In response to these limitations, an efficient data structure based on *k-ary* hash tables (Fig. 1), called sketch [1, 11, 12] was proposed and used to handle large state space, with a small amount of memory requirement and a linear computational (update/query) complexity. It is a multistage bloom filter based on random aggregations, where flow identifier (denoted by key) is mapped to index of bucket using *k* different hash functions (one index per stage of the array of *k* hash tables). These hash functions are generally chosen to reduce collision effect and to uniformly distribute keys over buckets of hash table.

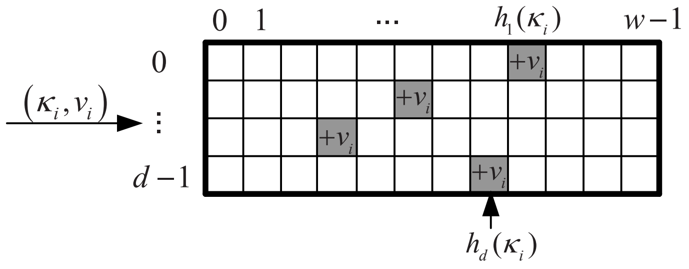


Fig. 1. Sketch data structure.

To use sketch in context of network anomalies detection, IP flows can be classified by some combinations of fields in packet header, such as destination IP address (*DIP*), or source and destination IP address (*SIP/DIP*), etc. This flow identifier is used as key to update the k^{th} hash table by its associated value (*key,value*). The value is a reward associated with key, and which can be the number of: packets, bytes, connection requests (*#SYN*), half open connections (*#SYN - #SYNACK*), *etc*.

Recent work with Count-Min Sketch (*CMS* [5]) has showed that random aggregation of flows does not significantly disrupt their variations. The *CMS* query request algorithm can indicate if a given key exhibits large accumulated value, and even one can query sketch data structure about an approximate estimation of occurrence frequency for a given key.

However, sketch is based on universal hash functions, which are not reversible. Consequently, we cannot use sketch to report the set of frequent or heavy hitter keys, because sketch does not store any information about its key entry. Thus, the only way to get all heavy keys (which exhibit heavy accumulated value) is to test all possible entry, by hashing for the second time all keys, in order to determine those mapped to heavy buckets. That mean when monitoring high speed links, all keys must be recorded and verified. Unfortunately, this approach is neither scalable nor accurate for online monitoring.

In this paper, we consider the problem of online detection of network anomalies over high speed links, in order to cope with attacks as soon as possible. We propose a new variation of sketch by adding an inversion procedure for sketch, and we use a parametric version of multi-channel CUSUM (M-CUSUM [10, 21, 22]) as sequential algorithm [9] for anomaly detection in each bucket of sketch. Our contribution is twofold. First, we resolve the problem of sketch inversion with a software compliant procedure method, and second we analyse the efficiency of M-CUSUM (a sequential algorithm for anomaly detection) over this compact way for storing flows information. Sketch was used for offline anomaly detection through searching heavy hitter flows, and the use of sequential change point detection algorithm over sketch was never been addressed.

The remainder of this paper is organized as follows. In the following section, we discuss the related work and previous research related to our work. In section III, we give a brief overview of *CMS* Sketch and parametric M-CUSUM mechanisms that are related to our studies. Section IV describes our proposed method for detecting change point in a reversible sketch. In section V, we present the analysis results from the application of the proposed framework over real internet traces. Finally, Section 6 presents concluding remarks and the future work.

2 Related Work

The authors of [24] use a non parametric version of CUSUM, as sequential hypothesis testing algorithm [9] for anomaly detection with TCP SYN flooding. In [20], the authors evaluate and compare two anomaly detection algorithms (adaptive threshold and CUSUM) for detecting TCP SYN attacks. They conclude that CUSUM is more efficient than adaptive threshold, especially for the detection of low intensity attacks. Moreover, they compare their results obtained with a parametric version of CUSUM (by assuming a normal distribution for SYN inter-arrival packets), with the result of non-parametric version used in [24], and they conclude to better performance. For this reason, we will restrict our study in this paper to the parametric version of CUSUM. However, CUSUM algorithm is not able to pinpoint the malicious flow responsible of anomaly. It only raises an alarm after the detection of anomaly. In [10, 21, 22], the authors prove that multichannel CUSUM is more efficient than single channel, which means applying CUSUM at flow level (given some criteria for packets aggregations and flows classifications), is more efficient in anomaly detection than applying this algorithm over the total number of ingoing packets.

The Multi-channel CUSUM (M-CUSUM [10, 21, 22]) is statistical self learning algorithm for initializing required parameters (e.g. mean and variance) in order to build an initial normal profile, in an adaptive manner with various network load and traffic patterns. M-CUSUM belongs to anomaly-based intrusion detection class, which detect a change in traffic parameters, through using the assumption that most anomalies induce a change in distributions of monitored parameters (mean, variance, etc.).

However, per flow application of CUSUM is prohibitive for real time operations over high speed networks, where storage and update operations of state-space flows information are very costly. In [1, 11, 12], a stretched data structure with a linear complexity of update/search operations, was proposed and used to handle large data. The authors in [3, 4, 13] have tackled the problem of offline anomaly detection over sketch by verifying if the values of the sketch buckets associated to a given key are heavy hitter or not. Recent work in [19] lookup for heavy buckets in the sketch resulted from the difference between current epoch and time series forecasting sketches. However, heavy hitter flows do not necessarily correspond to malicious flows. Therefore, we will address this problem by using M-CUSUM over sketch buckets in this paper.

Sketch is based on universal hash functions and random aggregations, and does not store information about active flows identifiers. In [12], it was used with the storage of all existing flows identifiers during a time interval, and through re-hashing of all stored identifiers to determine malicious flows. Unfortunately, storing all flows identifiers, especially when monitoring high speed links, is neither scalable nor efficient for online monitoring. There is a need for a software compliant reversal procedure over sketch to pinpoint corresponding keys to malicious flows.

3 Background

In this section, we briefly survey the underlying count-min sketch data structure and multi-channel CUSUM theory related to our work.

3.1 Count-Min Sketch

Let $S=s_1s_2...s_n$ be the set of input stream that arrives sequentially, item by item [5]. Each item $s_i=(\kappa_i, v_i)$ is identified by a key $\kappa_i \in U$ drawn from a fixed universe U of items. Each key κ_i is associated with a reward (or update for frequency occurrence) value $v_i \in R$. The arrival of item with key κ_i increment its associated counter in the j^{th} hash table by v_i ($C[j][h_j]$ += v_i), as shown Fig. 1. The update procedure is realized by d different hash function, chosen from the set of 2-universal hash function $H_j(\kappa_i)=((a_j\kappa_i + b_j)\ modP_U)\ modw$ to uniformly distribute κ_i over hash tables and to reduce collision. Parameter P_U is a prime number larger than the maximum number in universe, and Mersenne prime numbers of the form 2^i-1 are generally chosen for fast implementation.

The count-min query returns an estimate of the accumulated value for a given key, as the minimum value of d counter ($\hat{s}(\kappa_i)=\min_{0\leq j<d} \{C[j][\kappa_i]\}$).

Ongoing IP packets into an ISP can be classified as series of (κ_i, v_i), where κ_i can be the IP destination address (*DIP*), or any other fields in packets header, and the value v_i can be the number of *SYN* request. *CMS* query can estimate if a given DIP (key) is under *SYN* flooding attack by verifying the value of $\hat{s}(\kappa_i)$.

In CMS we use $d=\lceil ln(1/\delta)\rceil$ pairwise independent hash functions, where each one receives κ_i as parameter and returns a random integer in the range $w=[0,e/\varepsilon]$. ε is the error rate with probability less than δ. Thus, it maintains modest storage requirements of $O(ln(1/\delta) \times (1/\varepsilon))$ count cells.

3.2 Multi-Channel Cumulative Sum Algorithm

In contrast to the most widely used techniques with sketch for anomalies detection (heavy hitter), sketch can be used with various sophisticated sequential detection procedures [10, 20, 24] to uncover anomalies. In this paper, we will focus on multi-channel CUSUM algorithm, due to its low computational overhead and modest storage requirements.

In this section, we briefly review the sequential parametric multi-channel CUSUM [10, 20] algorithm used to detect change point in traffic. CUSUM relies on two phases: training and detection. In training phase, it establishes and updates a dynamic behavior profiles for normal flows. In detection phase, it uses log likelihood ratio to detect any kind of abrupt deviation from well established profile. In multi-channel version of CUSUM, the algorithm is applied over many channels, and once an anomaly is detected in any channel, an alarm is raised.

Let $\{X_{ij}^{nT}, 1 \le i \le d, 1 \le j \le w\}$ be the value of each bucket during the n^{th} time interval. Observations X_{ij}^{nT} are *i.i.d* with a *pdf* $f_{ij,\gamma_0}(x)$ for $n < t_a$ (before attack occurrence) and with another *pdf* $f_{ij,\gamma_1}(x)$ for $n \ge t_a$ (after attack), where t_a is the instant of attack detection. M-CUSUM tests statistical hypotheses H_{ij} (eq. (1)) to detect abrupt change in bucket with index (i,j) at the time epoch $n=t_a$:

$$H_{ij,0}: \quad \gamma_{ij} = \gamma_0 \qquad \text{Versus} \qquad H_{ij,1}: \quad \gamma_{ij} = \gamma_1 \tag{1}$$

Where γ_0 and γ_1 are respectively the *pdf* parameters before and after change occurrence. The detection of anomaly is based on log likelihood ratio for an observation X_{ij}^{nT} test between the two hypotheses:

$$s_{ij}^{nT} = \ln\left(\frac{pr(X_{ij}^{nT} \mid \gamma_1)}{pr(X_{ij}^{nT} \mid \gamma_0)}\right) \tag{2}$$

If we assume Gaussian distribution $f_{ij,\gamma_0}(x) = N(\mu_{ij,0}, \sigma_{ij,0}^2)$ for the hypothesis $H_{ij,0}$ and $f_{ij,\gamma_1}(x) = N(\mu_{ij,1}, \sigma_{ij,1}^2)$ for $H_{ij,1}$, the log likelihood ratio is then:

$$s_{ij}^{nT} = \ln \frac{\sigma_{ij,0}}{\sigma_{ij,1}} + \frac{(X_{ij}^{nT} - \mu_{ij,0})^2}{2\sigma_{ij,0}^2} - \frac{(X_{ij}^{nT} - \mu_{ij,1})^2}{2\sigma_{ij,1}^2} \tag{3}$$

The cumulative sum is a summation of the log likelihood ratio:

$$S_{ij}^{nT} = \sum_{k=1}^{n} s_{ij}^{kT} \tag{4}$$

S_{ij}^{nT} will increase when $s_{ij}^{kT} > 0$, and decreases for $s_{ij}^{kT} < 0$. When the value of S_{ij}^{nT} become greater than threshold h, a decision can be taken about the hypotheses ($H_{ij,0}^{nT}$ for normal condition and $H_{ij,1}^{nT}$ for attack condition). Therefore, the relevant information for detecting change lies in the difference between the value of the log-likelihood ratio and its current minimum value [20]. Hence the stopping time for the CUSUM algorithm is defined by:

$$t_a = t_a(h) = \min\{n \geq 1 : G_{ij}^{nT} \geq h\} \tag{5}$$

Where:

$$G_{ij}^{nT} = S_{ij}^{nT} - m_{ij}^{nT} \text{ and } m_{ij}^{nT} = \min_{\substack{1\leq i\leq d \\ 1\leq j\leq w}} S_{ij}^{nT} \tag{6}$$

The statistic function G_{ij}^{nT} obeys the recursion:

$$G_{ij}^{nT} = \max\left\{0, G_{ij}^{(n-1)T} + \ln\left(\frac{pr(X_{ij}^{nT} \mid \mu_1)}{pr(X_{ij}^{nT} \mid \mu_0)}\right)\right\} \text{ and } G_{ij}^{0} = 0 \tag{7}$$

We consider that X_{ij}^{nT} follows a Gaussian distribution with known variance σ_{ij}^2 that remains unchanged after the attack, and μ_0 and μ_1 are the mean before and after attack. μ_1 can be estimated online in self learning manner, under the condition that attack occurrence leads to change in the mean value of $\mu_0 \rightarrow \mu_1$ ($\mu_1 > \mu_0$ with a value of $\mu_1 = \alpha\mu_0$), by following the same analysis and assumptions in [20], where authors have conclude to more false alarm and more miss detection when replacing the Gaussian distribution assumption in CUSUM by its non-parametric version. It is important to recall that other statistical parameters may change with the mean value under attack, and they will be treated as noise. With the assumption of Gaussian distribution, eq. (7) takes the following forms:

$$G_{ij}^{nT} = \max\left\{0, G_{ij}^{(n-1)T} + \frac{\mu_{ij,1} - \mu_{ij,0}}{\sigma_{ij}^2}\left(X_{ij}^{nT} - \frac{\mu_{ij,1} + \mu_{ij,0}}{2}\right)\right\} \text{ and } G_{ij}^0 = 0 \tag{8}$$

After the substitution of $\mu_{ij,1}$ by $\mu_{ij,1}=\alpha\mu_{ij,0}$, eq. (8) becomes:

$$G_{ij}^{nT} = \max\left\{0, G_{ij}^{(n-1)T} + \frac{(\alpha-1)\mu_{ij,0}}{\sigma_{ij}^2}\left(X_{ij}^{nT} - \frac{(\alpha+1)\mu_{ij,0}}{2}\right)\right\} \text{ and } G_{ij}^0 = 0 \tag{9}$$

The detection of anomaly is given by testing the value of M-CUSUM function G_{ij}^{nT} (if $G_{ij}^{nT} > h$ then alarm is raised). The values of $\mu_{ij,0}$ and σ_{ij}^2 can be estimated in a self learning phase and updated dynamically using *EWMA* (Exponential Weighted Moving Average) formulas in eq. (10):

$$\begin{aligned} \mu_{ij,0}^{nT} &= \beta\mu_{ij,0}^{(n-1)T} + (1-\beta)X_{ij,0}^{nT} \\ \sigma_{ij}^2(nT) &= \beta\sigma_{ij}^2\left((n-1)T\right) + (1-\beta)(X_{ij,0}^{nT} - \mu_{ij,0}^{nT})^2 \end{aligned} \tag{10}$$

In fact, with the large fluctuations and variations in traffic characteristics (heavy tailed distributions for packets length, memory-less inter-arrival, self-similarity and long-range dependence, etc.), and with the lack of consensus about distributions of traffic characteristic parameters, one may wonder about the efficiency of Gaussian distribution assumption for CUSUM. The M-CUSUM [10] function in non parametric version is updated using the following formula:

$$G_{ij}^{nT} = \max\left\{0, G_{ij}^{(n-1)T} + c_{ij}(X_{ij}^{nT} - \mu_{ij,0} - \varepsilon_{ij}\mu_{ij,1})\right\} \text{ and } G_{ij}^0 = 0 \tag{11}$$

However, the problem is in choosing the parameters (c_{ij}, ε_{ij}, and $\mu_{ij,1}$) that control the sensitivity of attack detection, which is not a straightforward task, and left to user. c_{ij} a positive weight which is set to 1 in [24]. Parameter ε_{ij} is a tuning parameter chosen from [0,1] due to the average delay detection [10] which must be a positive number:

$$ADD_{t_0}(t_a) = \frac{h}{(1-\varepsilon_{ij})\mu_{ij,1} - \mu_{ij,0}} > 0 \Rightarrow \varepsilon_{ij} < 1 - \mu_{ij,0}/\mu_{ij,1} \tag{12}$$

In fact, we can get the Gaussian parametric version of CUSUM (given in eq. (8)) from the non-parametric version by substituting parameters ε_{ij} and c_{ij} in eq. (11) by:

$$c_{ij} = \frac{2\varepsilon_{ij}\mu_{ij,1}}{\sigma_{ij}^2} \text{ and } \varepsilon_{ij} = \frac{\mu_{ij,1} - \mu_{ij,0}}{2\mu_{ij,1}}. \tag{13}$$

Therefore, parametric version of CUSUM is not more than special case of non-parametric version, and choosing normal distribution becomes parameters adjustment choice for CUSUM detection algorithm.

A good detection procedure should have a low false alarm rate FAR and small values of the expected detection delay. In [22], it is proven that CUSUM is asymptotically optimal. It minimizes the average delay detection $ADD_{t_0}(t_a)$ for a fixed false alarm rate $\overline{FAR}$, i.e $FAR(t_a) \le \overline{FAR}$. The FAR increases by decreasing the speed of detection, and a trade-off between low FAR and minimum delay detection is required. The threshold value should be chosen from the condition $E_0 t_a(h) = 1/\overline{FAR}$ to minimize delay detection given a $\overline{FAR}$. It is worth noting that value of threshold h controls the sensitivity of the attack detection, hence large value of h decreases the FAR, but true attacks may also completely missed. We refer to [10, 20, 21, 22] for complete reference about CUSUM parameters and implementation details.

4 Proposed Approach

Our proposed framework is based on 2 data summary architecture: a Multi-Layer Reversible Sketch (*MLRS*) and a Count-Min Sketch (*CMS*) as shown in Fig. 2. The operations of the proposed framework are performed by two steps. Firstly, it continuously updates the two sketches (*MLRS* and *CMS*) counters from input data stream (κ_i, v_i) for a fixed time interval T. Secondly, it applies M-CUSUM in the background at each bucket to detect anomalies. Afterward we identify and output keys that mapped to buckets with a raised alarm by CUSUM.

In this paper, we seek to detect victim servers of TCP SYN flooding, as it was widely shown in the literature that more than 90% of the DoS attacks use TCP [15]. Ongoing packets are classified by DIP as key for flow identifier, and only packets with bit SYN set to 1 in TCP flag are considered. We associate with the key κ_i the DIP, and with v_i the value of bit SYN in packet header $((\kappa_i, v_i)=(DIP,SYN))$. We can also monitor another kind of flooding with TCP (ACK, RST, FIN, etc.) or with other protocol (flooding UDP, SMURF, etc.) by changing the associated reward v_i, but we will restrict our analysis to TCP SYN flooding in this paper.

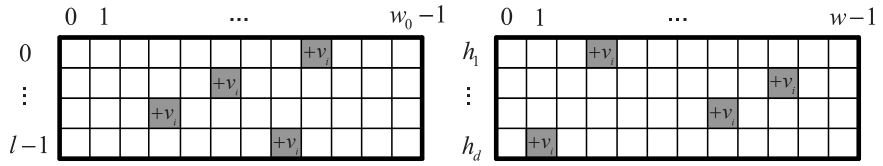


Fig. 2. MLRS and CMS sketch.

However, M-CUSUM only raises alarm in buckets after the detection of abrupt change, and due to random aggregations and collisions occurrences, reversing sketch is a difficult operation to uncover responsible flow of anomaly. There is only two existing approaches in the literature. The first [12] is based on intuitive idea by storing all keys in T time interval, and achieve verification by hashing for the second time the set of stored keys at the end of each interval. This strategy is inefficient in term of storage space and update speed for the list of keys.

The second approach [6, 19] is based on modular hashing and mangling via Galois Field $GF(2^n)$ operators, which is complex and more efficient for hardware implementation, as it was done in [19].

Our idea to reverse sketch is based on exploiting index in an additional multi-layer reversible sketch (Fig. 2), where indexes are used to store keys. In fact, the *MLRS* is used in the same way as *CMS* sketch, where the arrivals of each key increments its associated counter. However, each key has l counter (one by layer), where we split the key of N bit into $l \times w_0$ bit, with $w_0=2^P$, and $l=\lceil N/P \rceil$. P is the number of bits used to split the key, and w_0 is used as layer width in *MLRS*. The update procedure is summarized in Algorithm 1.

Algorithm 1 *Sketch Update procedure*

```
1:   $Mkey = encrypt\_optimized\_RC4(key);$
2:   $for\ i = 0\ to\ d-1\ do$
3:       $j = universal\_hash_i(Mkey);$
4:       $CMS[i][j].counter += v_i;$
5:   $end\ for$
6:   $for\ j = 0\ to\ l-1\ do$
7:       $MLRS[j][Mkey \,\&\, (2^P-1)].counter += v_i;$
8:       $Mkey \ >>= P$
9:   $end\ for$
```

If we seek to search for victims DIP (or keys that hash to buckets with raised alarm by M-CUSUM), we can release hierarchical search procedure in *MLRS*. If we don't find at least one bucket with raised alarm in each of the i^{th} ($i \leq l\text{-}1$) first layers of *MLRS*, there is no need to continue searching in other deep layers or through the second *CMS* sketch. Malicious flows must have one alarmed bucket in each layer.

We will begin by the simple case, where we assume that there is at most one bucket with CUSUM raised alarm in each layer as shown in Fig. 2. To recover key, we concatenate the l index in *MLRS* and we get the value of suspect key (e.g. *DIP*). We can not be sure of suspect before verification, where due to collision with other IP prefix, their value becomes large. The suspect key is verified through hashing and verification (by count-min query of CUSUM function) in the *CMS* for confirmation.

In general, even with a different value of width (e.g. 2^{12} or 2^{14}) for the *MLRS*, many buckets in different layers will be subject to collision occurrence, and in some case, we will be found with a bigger set of keys to verify through *CMS* than the original one. Nevertheless, it is important to notify that even if the set of suspect key is larger than departure one, it requires a small memory and fast update time with respect to original list.

To resolve this problem and reduce collision in *MLRS*, we use the idea of IP-mangling technique presented in [19], but with an optimized version of RC4 (Ron's Code [7]) ciphering algorithm rather than Galois Field $GF(2^n)$. The optimized RC4 code is available from [8].

IP mangling is a reversible procedure, which randomizes the input data in an attempt to destroy correlation between keys, and to disperse adjacent keys uniformly at all available buckets. Mangled key is denoted by Mkey in this paper. This technique is a bijective function that maps keys in a universe U to U. Each key κ_i is mapped to $y_i=f(\kappa_i)$, with the function f chosen in a way to destroy any correlation between keys, as show in Table 1. Any bijective function able to destroy correlation between keys, and return a completely random set of keys, can be used. Afterward, we use $f^{1}(y_i)$ to recover suspect key κ_i from *MLRS*.

Table 1. Mangling DIP by optimized RC4.

• DIP	• Mangled key
• 192.168.92.40	• 10010100101001011110100010011011
• 192.168.92.41	• 10101011011001000011001000100110
• 192.168.92.42	• 10010110111011000010010010101110
• 192.168.92.43	• 00100000001101001000000001101101

In [19], the function $f(\kappa_i) = a \otimes \kappa_i \oplus b$ is used for mangling, where $\otimes$ is the multiplication operator defined on $GF(2^n)$, $\oplus$ is the bit-wise XOR operator, and a and b are two random number uniformly chosen from the universe U. The reverse of a mangled key is obtained from $f^{-1}(y_i) = a^{-1} \otimes (y_i \oplus b)$ by precomputing the value of a^{-1}. In the other hand, the authors of [19] explain clearly that the direct calculation of $a \otimes x$ is very expensive, as it requires multiplying two polynomials (of degree n-1) modulo an irreducible polynomial (of degree n). Therefore, they use tabulation and many additional pre-computing tables to reduce complexity. In fact, Galois field is based on bit by bit operations which is hardware compliant, and requires additional memory for fast calculation of polynomial product. In contrast, the optimized RC4 bloc cipher algorithm is ideal for software implementation, as it requires only byte manipulations and its implementation is based on few lines of code. It has been proven to be powerful in our experimentations for mangling and destroying any correlation between adjacent keys, in terms of random Hamming distance between adjacent keys, as shown binary values in Table 1.

In Fig. 3, we show the distribution of collision when using direct mapping and mangling to update multi-layer sketch with $P=8$ for a universe size of 32-bit ($\kappa_i=DIP$). Data traces from 1 minute real bidirectional Internet traffic of 1776 flows (here flows are classified by DIP). In fact, used mangling techniques allow to uniformly distributing key over buckets, and prevent collision over IP with same prefix.

At the end of each time interval T, and after updating counters of *MLRS* and *CMS* continuously in online manner, M-CUSUM anomaly detection algorithm run in the background, to update CUSUM function in each bucket, and to raise alarm in bucket where the value of function G_{ij}^{nT} exceeds the threshold. Afterward, we scan *MLRS* for identification of all possible sequence of l bucket (one per layer) with triggered alarm and we realize verification through count-min query over the *CMS*, to ensure that the corresponding buckets with the d universal hash functions have a triggered alarm by CUSUM. However, we don't store the set of suspect keys, but once we have a suspect, we realize verification through the *CMS* before integrating it in alert message.

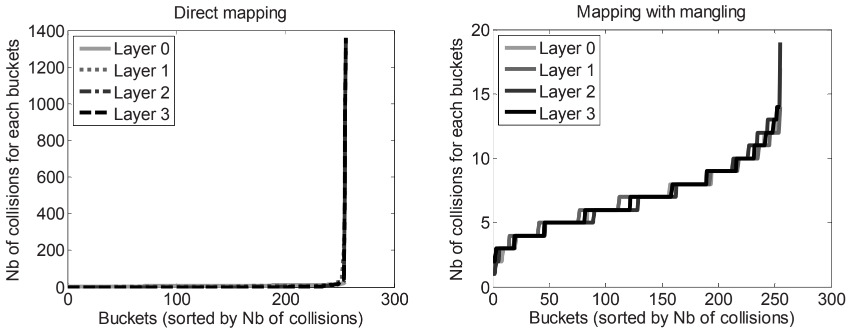


Fig. 3. Distribution of collisions for each bucket.

The hierarchical search procedure for alarmed buckets in *MLRS*, and the verification through *CMS* sketch are given in algorithm 2, for a universe of size 2^n, and a width of 2^P for *MLRS*, *P=n/2* and *l=2*. Boolean alarm variable is used to indicate the state of CUSUM function.

Algorithm 2 *Hierarchical search and verification*

```
1:   for $i = 0$ to $2^P - 1$ do
2:     if (MLRS[0][i].Alarm) then
3:       for $j = 0$ to $2^P - 1$ do
4:         if (MLRS[1][i].Alarm) then
5:           $Mkey = (j >> P) | i$;
6:           $Alarm = cms\_query(CMS, Mkey)$;
7:           if (Alarm) then
8:             $DIP = decrypt\_optimized\_RC4(Mkey)$;
9:             WriteLn(DIP);
10:          end if
11:        end if
12:      end for
13:    end if
14:  end for
```

5 Experiments Results

In this section, we present performance analysis results of juxtaposing M-CUSUM detection algorithm over reversible sketch, for detecting victims of TCP SYN flooding attacks. We have implemented M-CUSUM over sketch in C using the code of CMS available from [14]. We applied the proposed algorithm over many public traces (LBL-TCP-3, Abilene, Auckland, etc.) available from [16], and other traces used in OSCAR RNRT French Research project (OTIP, ADSL). Online implementation over Endace DAG 3.6ET is realized, and many experiments have been conducted for accuracy analysis. Our results are encouraging in terms of accuracy and response time.

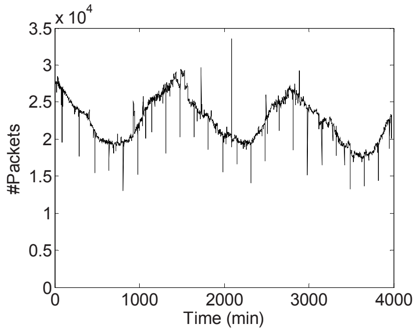
In this paper, we present the results of our experiments over the set of traces used in OSCAR project, and extensively studied by other partners in this project. We are interested in detecting victim of DoS/DDoS SYN Flooding in these traces. Afterward, we conduct performance analysis to study the influence of parameters at true positive and false detection.

The parameters we considered for the M-CUSUM algorithm were: threshold $h=5$, $\alpha=1.5$, $\beta=0.9$ as in [20]. For sketch parameters: $P=8$ unless otherwise noted, $w_0=256$, $l=4$, $d=4$ hash functions chosen from the set of 2-universal hash function, and with the use of tabulation [23].

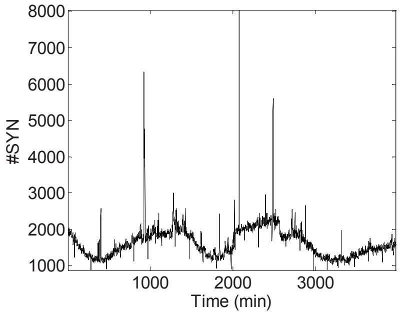
First, we present our analysis result over anonymized OTIP traces: 3 days of bidirectional traces collected by France Telecom ISP with Netflow format (~6.9GB) and contains ~896.10^5 flows.

Figure 4a and b show the variation of the number of packets during a time interval $T=1min$, as well as the variation of number of SYN during the 3 days. We have applied our proposed framework over this trace to uncover attacks, and we isolate the number of connection request received by each of identified victim as shown in Fig. 4c. The separation of received SYN by each destination is realized for additional information about the false alarm rate, and has been used for manual verification. The raised alarms by M-CUSUM for detected victims and their IP addresses are presented in Fig. 4d.

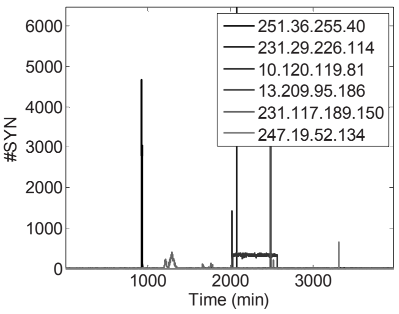
M-CUSUM detects a constant rate attacks as soon as it begins, as well as attacks with increasing rate, and with high sensitivity in detection of attacks with high/low intensity (Fig. 4c). However, in constant rate attacks, CUSUM raises alarm only in the beginning phase (the first few minutes) of flooding, where it updates the mean value by EWMA formula, with a slow rate ($1-\beta=0.1$) of convergence after change occurrence. It is worth noting that response time for analyzing the whole 3 days OTIP trace is less than 2 min. over a Pentium 1.72 Ghz with 1 GB of RAM memory.



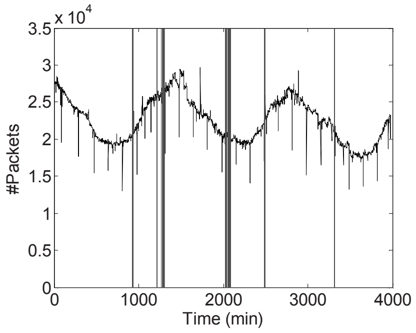
(a) # Packets



(b) #SYN



(c) Detected malicious flows

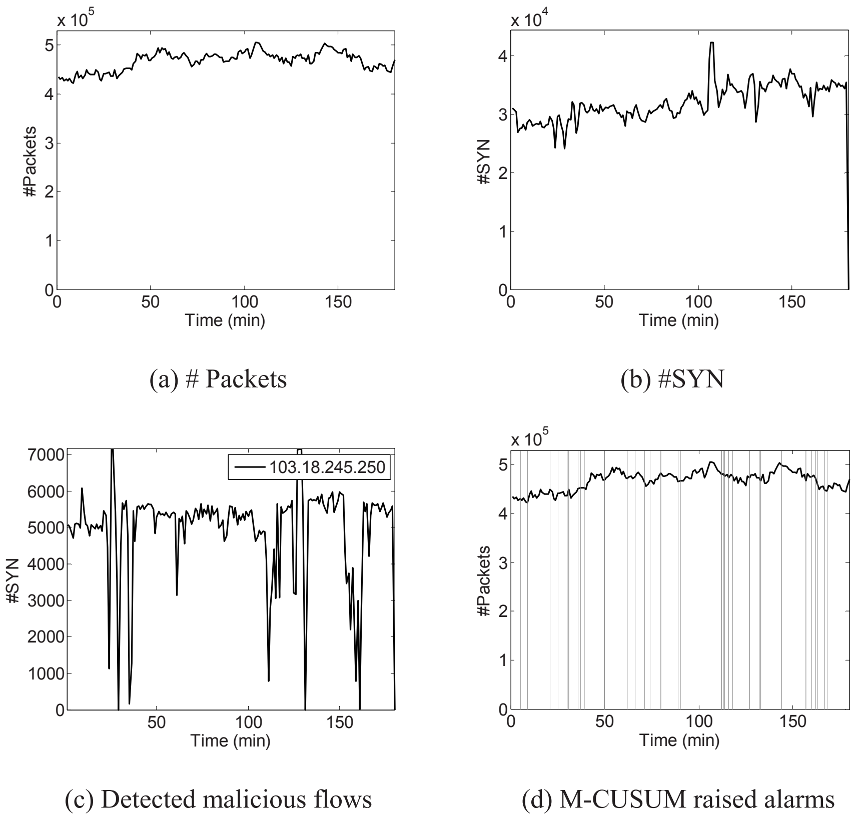


(d) *M-CUSUM raised alarms*

Fig. 4. Analysis results for OTIP trace.

Our second experiments consider two anonymized unidirectional traces ADSL (up and down) during 3 hours of capture in pcap format. We realize the same analysis study and manual verification as in the first experiment. First, we show the result obtained over upload traffic, and afterward we present the analysis result for the down traffic.

Figure 5a and b show the variation of the total number of packets and the number of SYN during a time interval of 1 min. Figure 5c and d show the number of SYN received by the only existing victims in this trace (with ~841.10^5 packets), and the raised alarms by M-CUSUM. Similarly, Fig. 6a and b show the total number of packets and the number of SYN in the down traffic (with ~825.10^5 packets), and Fig. 6c and Fig. 6d show the number of SYN received by the two detected victims and the raised alarms by CUSUM. However, deep manual investigations show that the DIP address (97.68.23.88) is not victim of SYN flooding, but of PortScan attack.

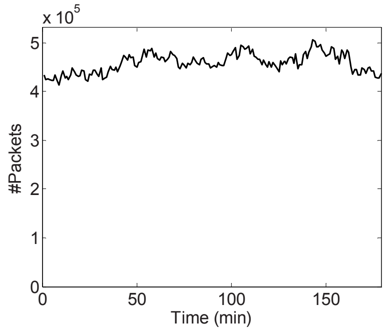


(a) # Packets (b) #SYN

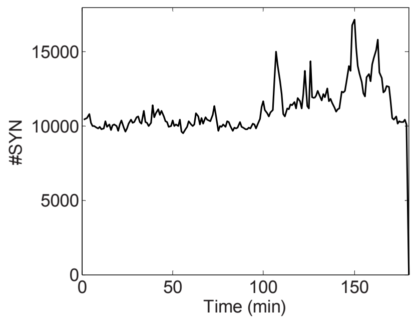
(c) Detected malicious flows (d) M-CUSUM raised alarms

Fig. 5. Analysis results for ADSL up trace.

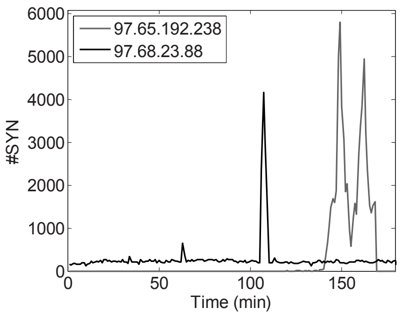
In the third set of our experiments, we conduct performance analysis study via Receiver Operational Characteristics (ROC) curve, to study the accuracy of the proposed framework. Our analysis verifies the false positive and true positive probabilities, with the variation of the value of threshold *h* and the *MLRS* sketch width. However, due to the lack of public well documented traces with well known attacks, we use the overall attacks uncovered by other research laboratory that have analyzed OTIP traces, as a complete set of existing one. Therefore, true positive and false positive probabilities are easily verified because we know in advance the IP address of victim servers, and the number of existing attacks. P_{TP} is the number of detected attacks divided by the total number of existing ones. P_{FP} is the percentage of raised alarm that did not correspond to real attack.



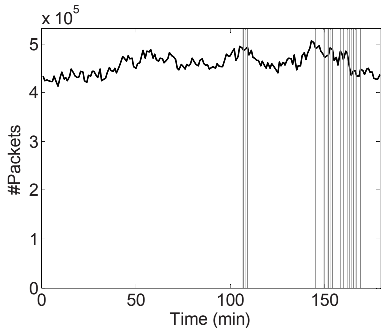
(a) *# Packets*



(b) *#SYN*



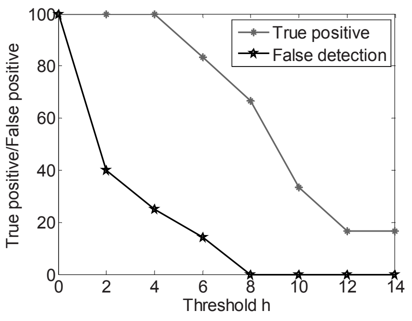
(c) *Detected malicious flows*



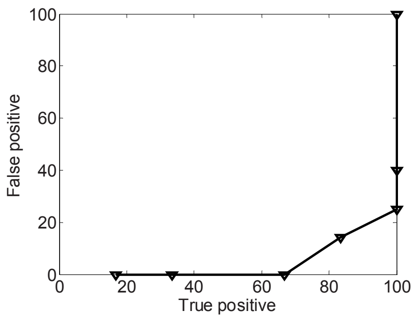
(d) *M-CUSUM raised alarms*

Fig. 6. Analysis results for ADSL down trace.

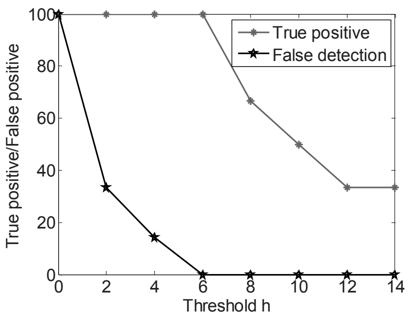
Figure 7 illustrates the relation between true positive and false positive ($P_{FP}=f(P_{TP})$), as well as $P_{FP}=f(h)$ and $P_{FP}=f(h)$, where false positive *FP* decreases as the threshold *h* increases, and true attacks may also completely missed. Hence, a tradeoff between false alarm and true positive detection is required to control the sensitivity and to prevent miss detection. We also notice that larger sketch width decreases the false positive and increases the detection rate. Furthermore, we investigate the effect of increasing the number of hash function at detection rate. Our analysis results show the existence of point after which increasing the number of hash functions does not improve the detection rate.



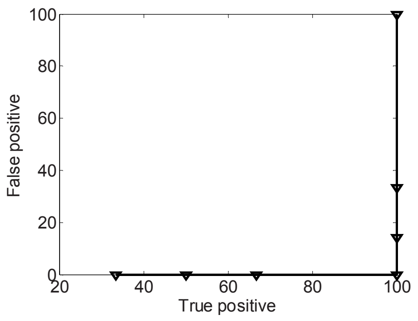
(a) P_{FP}*=f(h) and* P_{TP}*=f(h) for P=8*



(b) P_{FP}*=f(*P_{TP}*) for P=8*



(c) P_{FP}*=f(h) and* P_{TP}*=f(h) for P=14*



(d) P_{FP}*=f(*P_{TP}*) for P=14*

Fig. 7. P_{FP}=f(h) and P_{TP}=f(h) for P=8 and P=14.

6 Conclusions

In this paper, we propose a new framework that integrates sketch and M-CUSUM, for online anomalies detection over high speed links. Proposed framework is able to automatically pinpoint the malicious IP flows responsible of anomaly, through exploiting bucket index in an additional multi-layer sketch.

We proved the effectiveness of the proposed approach through implementation and testing on real traces with DoS/DDoS. Results of our experimentations have proved the capacity of early detection even for low intensity of DoS/DDoS attacks.

The proposed method is easily decentralized due to linear property of sketch with respect to addition operator. Ongoing work will be converged toward the hierarchical distribution of the proposed approach, and the reduction of the size of exchanged sketch information between different monitoring nodes in different layers.

Acknowledgments

This work has been partially funded by the French National Research Agency through the OSCAR project.

References

[1] Charikar M, Chen K, Farach-Colton M (2002) Finding frequent items in data streams. In: 29th International Colloquium on Automata, Languages and Programming (ICALP '02), London, UK, pp. 693–703.

[2] Cisco Systems Inc: Cisco netflow. http://www.cisco.com/wrap/public-/732/Tech/netflow

[3] Cormode G, Korn F, Muthukrishnan S, Srivastava D (2004) Diamond in the rough: Finding hierarchical heavy hitters in multi-dimensional data. In: 23rd ACM SIGMOD, pp. 155–166.

[4] Cormode G, Muthukrishnan S (2004) What's new: Finding significant differences in network data streams. In: IEEE Infocom'04, pp. 1534–1545.

[5] Cormode G, Muthukrishnan S (2005). An improved data stream summary: The count-min sketch and its applications. Journal of Algorithms, 55(1):58–75.

[6] Feng W, Zhang Z, Jia Z., Fu Z (2006). Reversible sketch based on the xor-based hashing. In: Asia-Pacific Conference on Services Computing (APSCC '06), Guangzhou, Guangdong, China, pp. 93–98.

[7] Fluhrer S, McGrew D (2001). Statistical analysis of the alleged RC4 keystream generator. In: 7th International Workshop on Fast Software Encryption (FSE '00), London, UK, pp. 19–30.

[8] Gutmann P (1996), Optimized RC4 code. http://www.zengl.net/freeswan/.

[9] Jung J, Paxson V, Berger A, Balakrishnan H (2004) Fast portscan detection using sequential hypothesis testing, in: IEEE Symposium on Security and Privacy, pp. 9–12.

[10] Kim H, Rozovskii B, Tartakovsky A (2004) A nonparametric multichart cusum test for rapid intrusion detection. Journal of Computing and Information Science, 2(3):149–158.

[11] Krishnamurthy B, Sen S, Zhang Y, Chen Y (2003) Sketch-based change detection: methods, evaluation, and applications. In: 3rd ACM SIGCOMM Conference on Internet Measurement (IMC'03), New York, USA, pp. 234–247.

[12] Li X, Bian F, Crovella M, Diot C, Govindan R, Iannaccon G, Lakhina A (2006) Detection and identification of network anomalies using sketch subspaces. In: 6th ACM SIGCOMM on Internet Measurement (IMC '06), New York, USA, pp. 147–152.

[13] Li Y, Yang J, An C, Zhang H (2007) Finding hierarchical heavy hitters in network measurement system. In: ACM Symposium on Applied Computing (SAC ’07), New York, USA, pp. 232–236.
[14] Massive Data Analysis Lab: MassDal: Count-min sketch source code. http://www.cs.rutgers.edu/7Emuthu/massdal-code-index.html
[15] Moore D, Voelker G, Savage S (2001) Inferring internet denial-of-service activity. In: Usenix Security Symposium, pp. 9–22.
[16] National Laboratory of Applied Network Research: NLANR: Traces archive. http://pma.nlanr.net/Special/.
[17] Paxson V (1999). Bro: A system for detecting network intruders in real-time. Journal of Computer Networks, 31(23–24):2435–2463.
[18] Roesch M (1999) Snort – lightweight intrusion detection for networks. In: USENIX Lisa ’99, Seattle, WA, USA.
[19] Schweller R, Li Z, Chen Y, Gao Y, Gupta A, Parsons E, Zhang Y, Dinda P, Kao M.-Y, Memik G (2006) Reverse hashing for high-speed network monitoring: Algorithms, evaluation, and applications. In: INFOCOM 06, pp. 1–12.
[20] Siris V. A, Papagalou F (2004) Application of anomaly detection algorithms for detecting Syn flooding attacks. In: GLOBECOM ’04, vol 4, Dallas, USA, pp. 2050–2054.
[21] Tartakovsky A (2005) Asymptotic performance of a multichart cusum test under false alarm probability constraint. In: 44th IEEE Conference on Decision and Control and the European Control Conference, Seville, Spain, pp. 320–325.
[22] Tartakovsky A, Rozovskii B, Blazek R, Kim H (2006) A novel approach to detection of intrusions in computer networks via adaptive sequential and batch-sequential change-point detection methods. Journal of IEEE Transactions on Signal Processing, 54(9):3372–3382.
[23] Thorup M, Zhang Y (2004) Tabulation based 4-universal hashing with applications to second moment estimation. In: ACM-SIAM Symposium on Discrete Algorithms (SODA ’04), New Orleans, LA, USA.
[24] Wang H, Zhang D, Shin K. G (2002) Syn-dog: Sniffing syn flooding sources. In: 22nd International Conference on Distributed Computing Systems (ICDCS’02), Washington, DC, USA, pp. 421–429.

5
Elastic Block Ciphers in Practice: Constructions and Modes of Encryption

Debra L. Cook[1, *], Moti Yung[2], and Angelos D. Keromytis[2]

[1]Columbia University, New York, USA dcook@cs.columbia.edu[1]
[2]Department of Computer Science, Columbia University, New York, USA
{moti, angelos}@cs.columbia.edu

Abstract. We demonstrate the general applicability of the elastic block cipher method by constructing examples from existing block ciphers: AES, Camellia, MISTY1 and RC6. An elastic block cipher is a variable-length block cipher created from an existing fixed-length block cipher. The elastic version supports any block size between one and two times that of the original block size. We compare the performance of the elastic versions to that of the original versions and evaluate the elastic versions using statistical tests measuring the randomness of the ciphertext. The benefit, in terms of an increased rate of encryption, of using an elastic block cipher varies based on the specific block cipher and implementation. In most cases, there is an advantage to using an elastic block cipher to encrypt blocks that are a few bytes longer than the original block length. The statistical test results indicate no obvious flaws in the method for constructing elastic block ciphers. We also use our examples to demonstrate the concept of a generic key schedule for block ciphers. In addition, we present ideas for new modes of encryption using the elastic block cipher construction.

1 Introduction

We illustrate the method for creating elastic block ciphers with four constructions. Elastic block ciphers are variable-length block ciphers created from existing block ciphers [5]. The elastic version of a block cipher supports any block size between one and two times that of the original block size. The method consists of a substitution-permutation network that uses

[*] This work was completed while the author was at Columbia University.

the round function from the existing fixed-length block cipher. In this work, we construct elastic block ciphers from AES [13], Camellia [1], MISTY1 [8] and RC6 [18], to serve as examples of the general applicability of the method. We analyze the randomness of the cipher's output using standard statistical tests and evaluate the performance of the elastic versions. We also use our constructions to illustrate the use of a generic key schedule for block ciphers. Additionally, we propose how the method can be used to create new modes of encryption.

Our performance tests demonstrate that the benefit of using an elastic block cipher varies based on the specific block cipher and implementation. In most cases, there is an increased rate of encryption when using an elastic block cipher to encrypt blocks a few bytes longer than the original block length as opposed to padding the data to two full blocks. The statistical tests applied to the block ciphers do not prove a cipher is secure but instead serve as a sanity check to determine if there are design flaws in the cipher. The test results for the elastic versions are consistent with those of the original ciphers and indicate no obvious flaws in the method for constructing elastic block ciphers.

The remainder of this paper is organized as follows. In Section 2, we describe our four constructions, including the use of a generic key schedule. In Section 3, we propose ideas for new modes of encryption. Section 4 concludes the paper.

2 Elastic Block Cipher Examples

2.1 Overview

We briefly review our method for creating elastic block ciphers [5]. Our method converts the encryption and decryption functions of any existing block cipher, *G*, that accepts blocks of size *b* bits to a variable-length block cipher, *G'*, that accepts block sizes of $b+y$ bits, where $0 \leq y \leq b$. Figure 1 shows the general structure of an elastic block cipher. The round function of *G'* is a cycle of *G*, where a cycle is the sequence in which all *b* bits have been processed by the round function of *G*. For example, in AES the round function is a cycle. If *G* is a Feistel network, a cycle is the sequence of applying the round function of *G* to the left and right halves of the *b* bit block. In each round of *G',* the leftmost *b* bits are processed by the round function and the rightmost *y* bits are omitted from the round function. Afterwards, the rightmost *y* bits are XORed with a subset of *y* bits from the leftmost *b* bits and the results swapped. What *y* bits are chosen from the leftmost *b* bits for use in the swap step may vary per round. The swap step

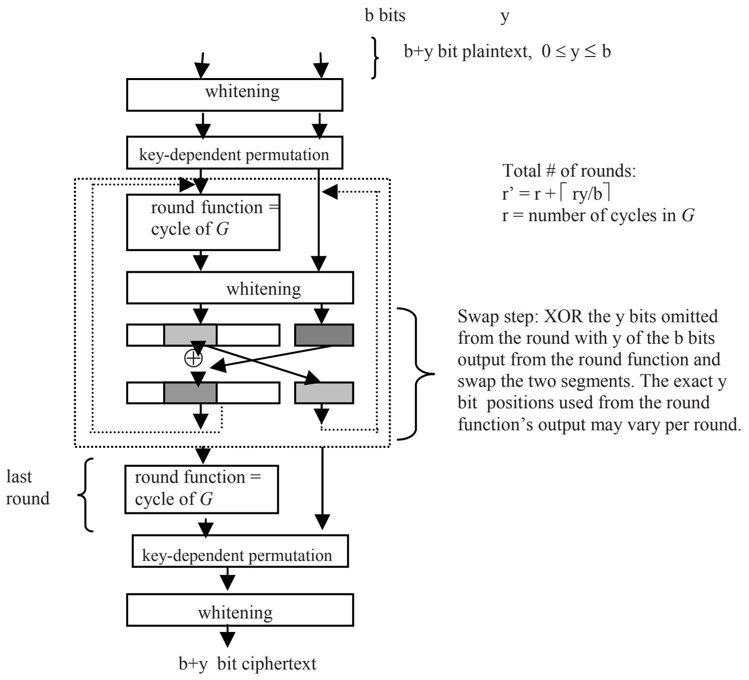

b bits
y
b+y bit plaintext, 0 ≤ y ≤ b
whitening
key-dependent permutation
Total # of rounds:
r' = r + ⌈ry/b⌉
r = number of cycles in G
round function = cycle of G
whitening
Swap step: XOR the y bits omitted from the round with y of the b bits output from the round function and swap the two segments. The exact y bit positions used from the round function's output may vary per round.
last round
round function = cycle of G
key-dependent permutation
whitening
b+y bit ciphertext

Fig. 1. Elastic block cipher structure.

is omitted after the last round. The number of rounds in G' is $r' = r + \lceil (ry)/b \rceil$ where r is the number of cycles in G. The elastic version also includes initial and end-of-round whitening on all b+y bits, and an initial and final key-dependent permutation that processes all $b+y$ bits.

In the remainder of this section we describe the elastic versions of AES, Camellia, MISTY1 and RC6. We choose these particular block ciphers because they were finalists in standards competitions that represent different methods for how the round function process bits. AES serves as the simplest example for creating an elastic block cipher because its round function processes the entire 128-bit block in each application. Camellia, one of the recommended 128-bit block ciphers from NESSIE's competition for cryptographic algorithms [10], is a Feistel network with an additional function applied after certain cycles. MISTY1, the recommended 64-bit block cipher from NESSIE, is also structured as a Feistel network. Its elastic version provides an example of a cipher covering blocks in the range of 64 to 128 bits. RC6, a finalist from the AES competition, breaks the data block into quarters and the round function updates two of the quarters using the values of the other two quarters. We use a 128-bit version of RC6.

2.2 Common Items

We first describe implementation details shared by the four examples. In the elastic versions of block ciphers, the bits in a block of data are numbered from the most significant (leftmost) to the least significant (rightmost). Bits 1 to b become the b-bit portion and bits $b+1$ to $b+y$ become the y-bit portion. The initial and final key-dependent permutations perform a byte or word level rotation combined with a swapping of any fractional byte of data. Two expanded-key bytes are utilized by each of the permutations. The amount of the rotation depends on an expanded-key byte. When the block size is not an integral number of bytes or words, the rightmost fractional byte or word is omitted from the rotation and swapped with bits from the rotation's result. A second expanded-key byte determines the byte or word from which bits are swapped with the fractional byte. If the block size is an integral number of bytes or words, this second expanded-key byte is unused. RC4 [17] was used for the key schedule. The first 512 bytes of RC4's output are discarded [9], then RC4 is run until the required amount of expanded key bytes are obtained. How the bits are selected for the swap steps varies slightly among our constructions. In all cases, the bits swapped out of the b-bit portion at the end of the round are y sequential bits (circling back to the leftmost bit after reaching the rightmost bit), but the starting position of this sequence varies per cipher. As shown in [4], the exact positions of the bits swapped does not matter in the sense that the elastic version will be secure against any attack that works by recovering key or round key bits if the original cipher is secure against the attack regardless of the bit positions chosen for each swap step.

For each cipher, we compared the performance of the elastic version to the original version with padding. We measured the rate of encryption for each block size that is an integral number of bytes. This excludes the time to expand the key. In the elastic implementations, when the block size is not an integral number of bytes, the fractional byte is stored in a byte and the processing time is the same as if a full byte of data is present; therefore, the time to encrypt $b+y$ bits is the time to encrypt $\lceil (b+y)/8 \rceil$ bytes. It is possible for the computational workload to vary at a more granular level, such as in a hardware implementation. The time for the fixed-length version to encrypt a $(b+y)$-bit block is the time to encrypt $2b$ bits in order to represent the padding required when using a b-bit block cipher. We measured the time to encrypt one million $(b+y)$-bit blocks, where $0 \leq y \leq b$ and y is an integer multiple of 8, using the elastic version and two million b-bit blocks using the fixed-length version. The time to pad the data was not included when measuring the performance of the original cipher. We implemented all the ciphers in C. All tests were conducted on a 2.8 Ghz

Pentium 4 processor with 1 GB RAM running Redhat Linux 2.4.22, unless otherwise noted.

We also compared the performance of the elastic versions to the performance of two previous proposals for variable-length block ciphers. The first proposal is by Bellare and Rogaway [2]. Their method involves running an existing block cipher, *G*, in CBC mode under one key, encrypting the last block of output from the CBC mode with *G* using a second key and using its output as an IV into *G* run in counter mode using third key. The ciphertext is the IV for the counter mode concatenated with the result of XORing the output from counter mode with the plaintext minus the last block. The second proposal is a modification by Patel, Ramzan and Sundarama to the first method that replaces the CBC portion with a hash function [15]. We used SHA-256 [14] as the hash function. Both proposals are less efficient than padding the plaintext to two full blocks and encrypting with a fixed-length block cipher, and both do not vary the workload for plaintext that is between one and two blocks in length. Bellare and Rogaway's method requires slightly more than twice the work of using fixed-sized, *b*-bit blocks for any $(b+y)$-bit block, where $0 < y \leq b$. Patel's method requires two full applications of the block cipher plus the cost of a hash function to encrypt $b+y$ bits.

2.3 Elastic AES

We created the elastic version of AES by adding the swap step between rounds of AES, expanding AES's whitening steps (AddRoundKey) from $b = 128$ bits to $128+y$ bits, and adding the initial and final key-dependent permutations. The round function consists of AES's SubBytes, Shiftrows and MixColumns steps, with the MixColumns step omitted in the last round to be consistent with the fixed-length version of AES [13]. The number of rounds ranges from 10 when $y = 0$ to 20 when $116 \leq y \leq 128$. We implemented the swap step by selecting *y* sequential bits from the leftmost *b* bits, wrapping around from the right to the left as needed. The starting position is varied by moving one byte to the right each round to avoid using the same bit positions in each swap. This avoids any complex selection process for choosing the *y* bits that would decrease performance.

We implemented two elastic versions of AES that differed in how the round function was implemented. In Version I, we implemented the round function as a straightforward sequence of the SubBytes, Shiftrows and MixColumns steps as defined in [13]. In Version II, we combined these steps into a table lookup. This results in the round function being a series of byte-level table lookups and XORs. Version II requires fewer CPU cycles than Version I, at the cost of an increase in memory usage. The round function can also be implemented to process the data as 32-bit words, in which case the table entries are 32-bit words. We kept table lookups at the byte level because we chose to implement the key-dependent permutations and swap step at the byte level.

The elastic versions increase the number of operations beyond the 128-bit versions due to the swap steps, the two key-dependent permutations and the expansion of whitening to cover *128 + y* its. In Version I, the elastic version saves processing time over padding. Obviously, as the block size approaches two full blocks, 20 rounds of AES are incurred in the elastic version along with the added steps, which increases the number of operations beyond the 20 rounds of AES that are required when padding the data to two full blocks. Therefore, it is expected that there is no performance benefit when encrypting blocks just under 32 bytes. In Version II, the elastic version does not offer a performance benefit compared to padding. This is because of the simplistic nature of the operations involved (table lookups and XORs) for the round function. Even though there are fewer rounds in the elastic version than with padding, the operations for the swap step and the two key-dependent permutations consume any savings gained from having fewer rounds. However, Version II offers a performance benefit over the variable-length block cipher construction by Bellare and Rogaway, and its modification by Patel, et al.

Figure 2 summarizes the results from the following three cases: Case 1: Version I tested on a 1.3 Ghz Pentium 4 processor with 512 MB RAM running Windows XP, Case 2: Version I tested in the Linux environment described in Section 2.2. Case 3: Version II tested in the Linux environment

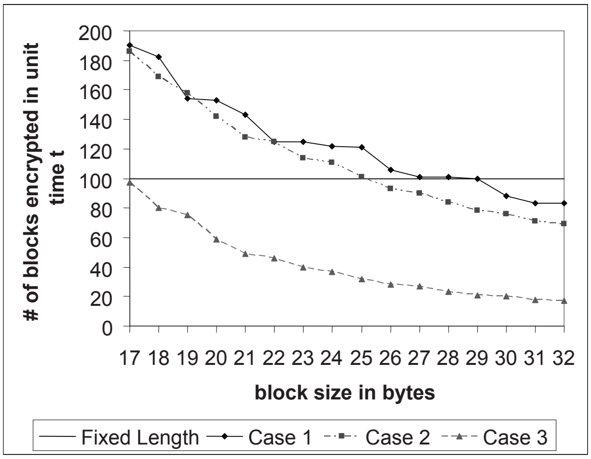


Fig. 2. Normalized # of blocks encrypted by elastic AES in unit time (regular AES = 100)}.

described in Section 2.2. In the first trial, the number of *(b+y)*-bit blocks the elastic version can encrypt per second ranges from 190% of the number of *2b*-bit blocks AES can encrypt per second when $y = 1$ to 100% when $y = 97$. Then the elastic version's performance decreased gradually to a low of 83% of AES's rate. In the second trial, the values ranged from 186% to 69% of AES's rate, with the elastic version becoming slower than the fixed-length version when $y = 73$. In the third trial, the elastic version was slower than the fixed-sized version with padding for all block sizes.

We compared Bellare and Rogaway's method and Patel's method to AES with padding on the Pentium 4 processor used in cases 2 and 3. Bellare and Rogaway's method encrypted between 49 and 50 *(b+y)*-bit blocks in the same amount of time AES with padding encrypted 100 blocks, for both Version I and II of AES. Patel's method encrypted 96 *(b+y)*-bit blocks in the time it took Version I of AES to encrypt 100 blocks, and encrypted 18 *(b+y)*-bit blocks in the time it took Version II of AES to encrypt 100 blocks. When using Version I, elastic AES is computationally more efficient than both Bellare and Rogaway's method and Patel's method for all block sizes. When using Version II, elastic AES is computationally more efficient than Bellare and Rogaway's method for block sizes up to 21 bytes in length, and is more efficient than Patel's method for block sizes less than 31 bytes and is as efficient as Patel's method for block sizes between 31 and 32 bytes.

2.4 Elastic Camellia

Camellia processes 128-bit blocks and is a Feistel network with additional steps. A function, referred to as the *FL* function, is applied after every three cycles in the Feistel network, except after the last three cycles. *FL* is applied to the left half and its inverse is applied to right half of the $b = 128$ bits. Camellia contains initial and final whitening steps, but not end-of-round whitening. Creating the elastic version involved using a cycle from the Feistel network as the round function, expanding the two existing whitening steps to cover *128+y* bits and adding end-of-round whitening steps to all the other rounds, and adding the same initial and final key-dependent permutations that we used in elastic AES. We apply the *FL* function after every three rounds, except for the last round. A round of the elastic version is shown in Fig. 3. The data is processed as bytes. The swap step was implemented by altering the starting positions between the left and right halves of the *b*-bit portion then rotating it one byte to the right within the half. Camellia has 9 cycles. The number of rounds in the elastic version ranges from 9 when $y = 0$ to 18 when $114 \leq y \leq 128$.

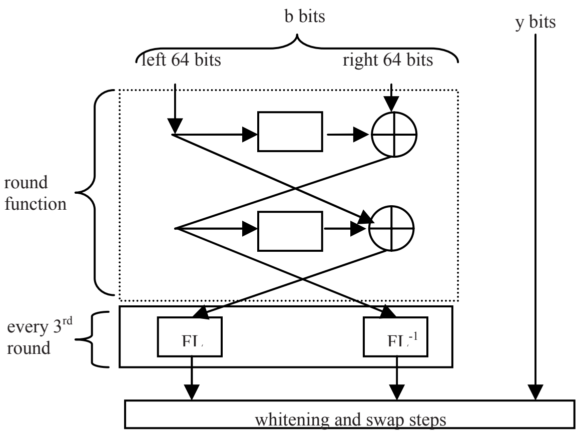


Fig. 3. Round function for elastic camellia.

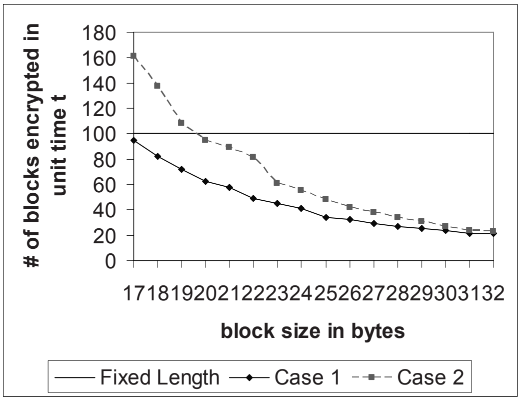


Fig. 4. Normalized #of blocks encrypted by elastic Camellia in unit time (regular Camellia = 100)}.

The elastic version offered no performance gain over the fixed-length version with padding. We also measured the performance of the elastic version without the initial and final permutations. Removing these two steps results in the elastic version offering a performance benefit when encrypting blocks that are one to three bytes over the normal 16-byte block size. Results for the following two cases are shown in Fig. 4: Case 1: elastic Camellia with all steps, Case 2: elastic Camellia without the initial and final key-dependent permutations. By using a lower bound of twice the work of padding for Bellare and Rogaway's method, elastic Camellia with the key-dependent permutations provides a performance benefit for block sizes up to 22 bytes and the version without the key-dependent permutations provides a performance benefit for block sizes in the range of 9–25 bytes compared to Bellare and Rogaway's method. Patel's method encrypted 61 *(b+y)*-bit blocks, $0 < y \leq b$, in the time it took Camellia with padding to encrypt 100 blocks. Elastic Camellia is more efficient than Patel's method for block sizes up to 21 bytes and 23 bytes, respectively, for the two cases.

2.5 Elastic MISTY1

MISTY1 is a 64-bit block cipher structured as a Feistel network with an additional function, called the *FL* function (not to be confused with the *FL* function from Camellia), applied once per cycle. While the number of cycles is not fixed, four cycles are recommended [10] and is the number upon which we base the number of rounds in the elastic version. MISTY1 does not contain whitening steps. A cycle from MISTY1 is used as the round function in the elastic version, shown in Fig. 5. Creating the elastic version involved adding the whitening steps, the initial and final key-dependent permutations and the swapping of bits after each cycle. The data is processed as 32-bit words. The key-dependent permutations are of the same form (a rotation and swap) as those used in the other three elastic block cipher examples. We alternate the starting position for the swap between the left and right halves of the round's output and, within each halve, rotate the starting position one word each time.

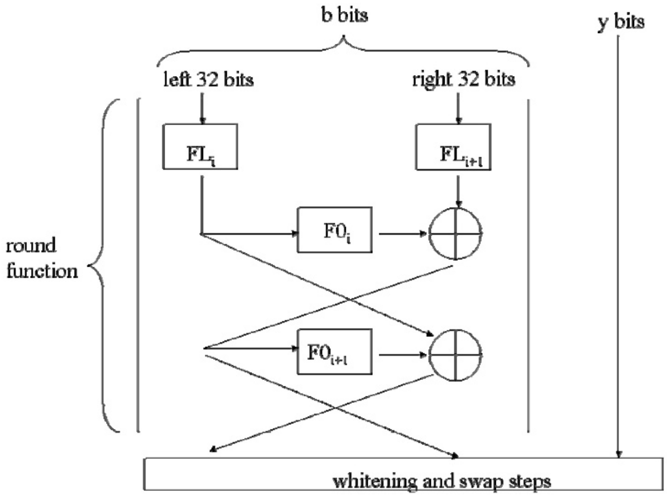


Fig. 5. Round function for elastic MISTY1.

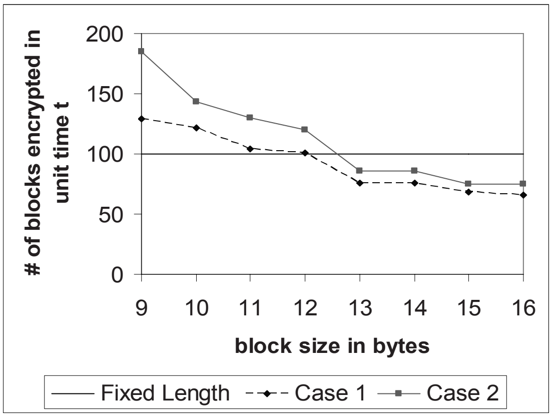


Fig. 6. Normalized # of blocks encrypted by elastic MISTY1 in unit time (Regular MISTY1 = 100).

We implemented elastic versions, with and without the key-dependent permutations, and the regular version of MISTY1. The performance results are shown in Fig. 6. Case 1 refers to the version with the key-dependent permutations and Case 2 refers to the version without the key-dependent permutations. The elastic versions increased the number of operations beyond the 64-bit version of MISTY1 due to the whitening, the swap steps and, in one version, the key-dependent permutations. The elastic version of MISTY1 provides a performance benefit compared to padding for blocks that are one to four bytes over the 8-byte block size that MISTY1 processes. The benefit increases significantly in Case 2 compared to Case 1 for block sizes that are up to one additional byte over MISTY1's 8-byte block size. The performance benefit from removing the initial and final key permutations decreases as the block size increases because they represent an increasingly smaller portion of the operations as more rounds are added. In both cases, the elastic version provides a performance benefit when compared to Bellare and Rogaway's method based on a lower bound of twice the work of padding for their method. Patel's method encrypted 51 $(b+y)$-bit blocks, $0 < y \leq b$, in the time it took MISTY1 with padding to encrypt 100 blocks using padding. Both cases of the elastic version of MISTY1 encrypt at a faster rate than Patel's method for all block sizes between 8 and 16 bytes.

2.6 Elastic RC6

RC6 is an example of a block cipher other than a Feistel network whose round function processes only a segment of the data block. RC6 divides a 128-bit data block into four 32-bit words, which we will refer to as ABCD. A and C are updated by the round function based on the values of B and D.

At the end of the round, A and C have expanded-key bits added to them then all the words are rotated to the left one word. B and D have expanded-key bits added to them before the first round, and A and C have expanded-key bits added to them after the last round. The addition of expanded-key bits to a word is a type of whitening. Since this "whitening" does not cover the entire data block and is not the same as performing whitening by XORing data with expanded-key bits, we view this addition as a step in the round function and not as whitening that should be expanded to all $b+y$ bits when forming the elastic version. A sequence of four applications of the round function of RC6 is a cycle and serves as the round function in the elastic version, as shown in Fig. 7. Initial and end-of-round whitening, and the initial and final key-dependent permutations are also added to create the elastic version. The rotations and XOR in the initial and final permutations were performed at the word level this time instead of at the byte level as done in elastic AES and elastic Camellia. The number of cycles in RC6 for 128-bit blocks is 5 (20 applications of RC6's round function). The number of rounds in the elastic version ranges from 5 when $y=0$ to 10 when $y=103$ (20–40 applications of RC6's round function). The swap step was implemented with the starting position rotating to the right one word each round.

The elastic version provides a performance benefit compared to padding for blocks of under 21 bytes in length. The results shown in Fig. 8. Using a lower bound of twice the work of padding for Bellare and Rogaway's method, the elastic version of RC6 provides a performance benefit for blocks under 30 bytes in length when compared to Bellare and Rogaway's method. Patel's method encrypted 52 blocks $(b+y)$-bit blocks, $0 < y \leq b$, in the time it took RC6 with padding to encrypt 100 blocks. Elastic RC6 is more efficient than Patel's method for block sizes up to 29 bytes.

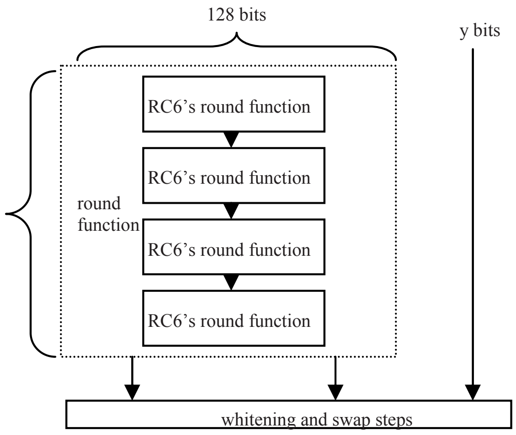


Fig. 7. Round of elastic RC6.

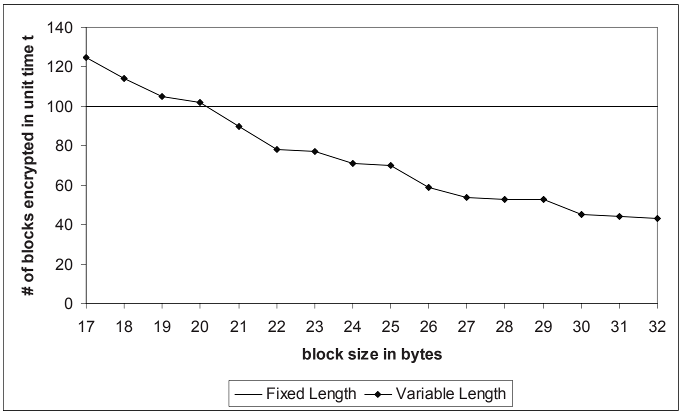


Fig. 8. Normalized # of blocks encrypted by elastic RC6 in unit time (Regular RC6 = 100).

2.7 Randomness Test Results

We applied statistical tests used by NIST on the AES candidates to both the original and elastic versions of the four ciphers. While these tests do not prove a cipher is secure, they do assist in determining if there are any obvious weaknesses with the cipher. There are sixteen tests performed on eight sets of data for each cipher. Refer to NIST's special publication 800-22 [12] for a description of the tests and to the NIST report entitled "Randomness Testing of the Advanced Encryption Standard Finalist Candidates" [11] for a description of the data sets. We tested every *(b+y)*-bit block size where y is an integral of 8 and $b \leq b+y \leq 2b$. We also tested two block sizes that were not an integral number of bytes. These were 129-bit and 171-bit blocks for the elastic versions of AES, Camellia and RC6, and 69-bit and 75-bit blocks for the elastic version of MISTY1. We used 128-bit keys in all of our tests. Each data set required either an initial set of random plaintexts or random keys. We created these random bit strings by extracting bits from files of random bits available from random.org [16]. Based on the results, each of our three elastic block cipher examples show no signs of any statistical weakness compared to the original ciphers. In the AES competition, finalists passed each test at a rate of 96.33% or higher [11]. The elastic versions of the ciphers also met or exceeded this rate. For the elastic versions of the ciphers, the percentage of samples passing each test was consistent across all block sizes and data sets.

2.8 Key Schedules

The key schedule for an elastic version of a block cipher has to generate more expanded-key bits than the key schedule of the original block cipher. Additional key bits are needed due to the expansion or addition of whitening steps, the two key-dependent mixing steps and the increase in the number of rounds. In practice, every block cipher includes its own key schedule, which is typically designed with a focus on performance and little concern about the lack of pseudorandomness in the expanded-key bits. This tendency in key schedule design results in key schedules contributing to attacks (due to the ease in which additional key bits can be determined once a few are found and by increasing the opportunity for related key attacks [3]) and forces applications supporting multiple block ciphers to support a separate key schedule for each cipher. When creating elastic block ciphers, we wanted to avoid these disadvantages of existing key schedules. Furthermore, unlike the encryption algorithms of block ciphers which follow a somewhat generic structure by being a series of rounds, key schedules vary extensively in their structures. This makes it unlikely a general method can be devised for modifying the key schedules to generate additional bits as needed based on the block size. Therefore, we required a generic key schedule that is independent of the block cipher and that generates as many pseudorandom expanded-key bits (or close to pseudorandom) as needed while adhering to a performance bound. Existing stream ciphers are potential candidates for satisfying these requirements. We used RC4 as the key schedule in the elastic block ciphers to illustrate the concept of a generic key schedule satisfying these requirements. The first 512 bytes of RC4's output are discarded due to a slight statistical weakness in the initial bytes output from RC4 [9]. We re-initialized RC4's "S" array is for each expanded key. A disadvantage of a generic key schedule is that if a weakness is discovered in the key schedule, it will impact any block cipher using the key schedule. However, having one key schedule decreases the likeliness of overlooked design flaws and implementation errors compared to when multiple key schedules are required.

In contrast to RC4 and any other stream cipher used in practice, the key schedules of AES and Camellia generate expanded keys that can easily be distinguished from random bits. In AES, an expanded-key byte is a combination of two other expanded-key bytes. When designing AES, Daemen and Rijmen noted the benefit of pseudorandom key bits, but stated that they took a "less ambitious" approach focused on avoiding symmetry between rounds and attacks due to related keys because "All other attacks are supposed to be prevented by the rounds of the block cipher." [6], page 77. In Camellia, there is a large overlap amongst the round keys. In MISTY1, the same expanded key bits are used in multiple locations within the block cipher. In RC6, it is more difficult to determine key bits from other expanded-key bits compared to AES and Camellia. Each original key byte is altered with an addition and a rotation. The resulting byte is then added to a previous expanded-key byte and a constant to create the next expanded-key byte.

We compared the performance of RC4 when generating enough expanded key bits to encrypt a *b*-bit block to the performance of the four ciphers' key schedules. When encrypting *b* bits, the number of expanded-key bits in an elastic block cipher is 32 more than the number in the original cipher (due to the key-dependent permutations) plus the number of bits needed for any initial and/or end-of-round whitening that was not in the original cipher. Recall that whitening steps were added when forming the elastic versions of Camellia and RC6; whereas, AES already contained whitening and only required that its whitening steps be expanded to cover all $b+y$ bits.

When measuring the performance of the original key schedules, we removed any statements from the original ciphers' key schedules that were present only for the support of key sizes other than 128 bits. Specifically, we removed the statements from AES's key schedules that were for the support of 192 and 256-bit keys. We also compared each elastic block cipher's key expansion rate to that of AES's original key schedule because in practice AES's key expansion rate is presently accepted. Let t_i, for i = 1,2,3,4, correspond to the key expansion rate for the fixed-length versions of AES, Camellia, MISTY1 and RC6, respectively. Table 1 shows the number of expanded-key bytes needed in the elastic block ciphers for block sizes of *b*, *b+8* and *2b* bits. The key-expansion rates for the elastic versions compared to that of the original versions are shown in Table 2.

Table 1. Number of expanded key bytes in elastic versions.

Cipher	Block size in bytes	# of rounds	# of Expanded-key bytes
AES	16	10	180
AES	17	11	208
AES	32	20	676
Camellia	16	9	340
Camellia	17	10	383
Camellia	32	18	980
MISTY1	8	4	196
MISTY1	9	5	246
MISTY1	16	8	444
RC6	16	20	516
RC6	17	21	562
RC6	32	40	1652

Table 2. Key expansion rate.

Elastic cipher	Block size in bytes	Elastic version's rate (RC4) vs fixed-length version's rate	Elastic version's rate (RC4) vs fixed-length AES's rate
AES	16	$5.94t_1$	$5.94t_1$
Camellia	16	$43.54t_2$	$6.89t_1$
MISTY1	8	$119.24t_3$	$6.09t_1$
RC6	16	$6.29t_4$	$7.84t_1$

We note that Camellia and MISTY1 have the fastest key schedule of the four ciphers and also requires the most expanded-key bits, thus resulting in RC4 appearing to be significantly slower. However, Camellia's and MISTY1's key schedules have the least amount of randomness of the four ciphers due to reusing expanded-key bits in multiple locations. Overall, the RC4-based key expansion used in the elastic ciphers when encrypting *b*-bit blocks is just under six to just under eight times the rate of AES's key schedule.

3 Modes of Encryption

3.1 Overview

An elastic block cipher can be used in existing modes of encryption in two ways. The first option is to use the block size of the original, fixed-length block cipher for all blocks except the last block, then use a variable-length block at the end to avoid padding. A second option is to use a block size different from the fixed-length block cipher for all blocks, with the size of the last block set to avoid padding. When using an existing mode, the only benefit the elastic version of a cipher provides is the elimination of padding; it does not eliminate any existing attack against the mode. For short segments of data between one and two blocks, an elastic block cipher allows all of the bits to be encrypted as a single block, avoiding the need to use a mode of encryption and creating a stronger binding across the ciphertext bits compared to the ciphertext produced by a mode of encryption. Elastic block ciphers also allow for new modes of encryption. We provide a sketch of two new modes, Elastic Chaining and Elastic Electronic Code Book (Elastic ECB). Both modes are intended as initial ideas for future work.

3.2 Elastic Chaining Mode

Elastic Chaining is depicted in Fig. 9. y bits from the i^{th} ciphertext block are prepended to the $(i+1)^{st}$ plaintext block and the result encrypted as a $(b+y)$-bit block. This concatenation creates a stronger binding between the i^{th} and $(i+1)^{st}$ blocks compared to that created by the XOR used in CBC mode. The stronger binding is achieved by increasing the work per block from the number of rounds required for b bits to the number required for $b+y$ bits, while the number of blocks is unchanged. The output consists of the leftmost b bits from each ciphertext block for all but the last block and the entire ciphertext of the last block. The first block to be encrypted can consist of b plaintext bits with a y-bit IV prepended to it, $b+y$ plaintext bits, or contain only b plaintext bits. Overall, the ciphertext will be at most y bits longer than the plaintext. If the plaintext is not an integral number of b-bit blocks, the last block may be shorter than $b+y$ bits. When the plaintext is not an integral number of b-bit blocks, the mode can be implemented without padding the last block; whereas, using the non-elastic version of the block cipher would require padding and also produce a ciphertext longer than the plaintext. The performance of the mode depends on the size of y. For a block cipher with r rounds, nr rounds are computed to encrypt n b-bit blocks with ECB, CBC or CTR mode. The number of rounds using elastic chaining will range from $n(r+1)$ when $y=1$ to $2nr$ when $\lceil (ry)/b \rceil = r$. This mode is useful in applications where the decryption can start at the last block. For example, when decrypting a file or segments of a database.

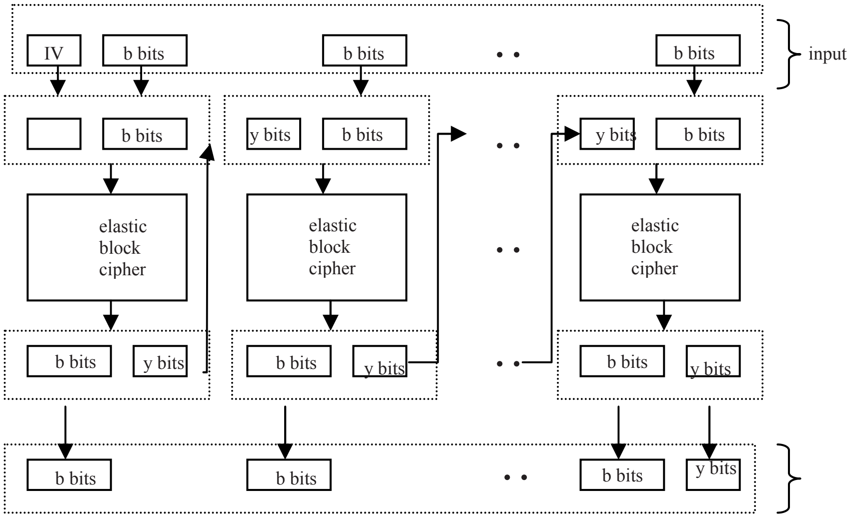


Fig. 9. Elastic chaining mode.

The ciphertext can be decrypted by decrypting the last block, concatenating the *y* bits from the plaintext block with the previous ciphertext block, and then decrypting the next block. When using an IV with the first block, the IV is not needed for decryption; however, having it available for decryption provides a type of integrity check in that the first *y* bits of the resulting plaintext can be verified against the IV.

The mode allows for variations. These include altering which positions the *y* bits from the previous ciphertext block are inserted into in the current plaintext block. Instead of prepending the *y* bits to the next plaintext block, they could be appended or inserted amongst the *b* bits as either *y* consecutive or nonconsecutive bits. The size of *y* can also vary between blocks, possibly based on the key value.

This mode offers several security benefits because, even if the plaintext is known, an attacker does not know the actual *(b+y)*-bit block being encrypted. If *y* varies per block based on key material, the attacker does not even know the length of each block being encrypted. Incorporating the previous ciphertext block into the current plaintext block when encrypting will hide plaintext patterns. In the way the mode is depicted in Figure 9, a single bit toggled in the ciphertext is detectable because it will garble all plaintext prior to and including the altered block. In order to insert or splice together ciphertext blocks, the inserted ciphertext block must decrypt to a plaintext which produces the same leftmost y bits as the original ciphertext block; otherwise, all plaintext blocks prior to this one will be garbled, resulting in a much more noticeable impact than the single garbled block produced by a splicing attack on CBC. Block-wise adaptive attacks [7], to which CBC is subject, are prevented because there is no need for the device performing the encryption to output the last y bits of each ciphertext block, except for the last block. This prevents the attacker from knowing the actual block being encrypted because the attacker only gets to choose *b* bits of the *b+y* bit block and block-wise adaptive attacks depend on the attacker knowing the exact plaintext.

To prepend blocks to the ciphertext, the attacker must be able to insert a ciphertext block that, when prepended to the leftmost y bits of the original first plaintext block will decrypt to some meaningful plaintext. Since these *y* bits are the IV, if the IV is not secret, the attacker will know what the *y* bits are and needs to find *b* bits that can be prepended to the *y* bits. However, notice that the attacker does not have a library of (plaintext, ciphertext) pairs from which to search for a possible *b*-bit value to prepend to the IV unless the entire plaintext is one block, in which case the mode is not necessary. The attacker will not have *(b+y)*-bit (plaintext, ciphertext) pairs from the (input, output) pairs of data encrypted with this mode because the leftmost *y* bits of the ciphertext are not included in the output except for the last block and the *b+y* input to the last block is not known. Appending blocks requires that the attacker append blocks of ciphertext which decrypt to a plaintext whose leftmost y bits are the same as the last *y* bits of the original ciphertext. In both cases, the smaller *y* is, the more likely it is that the attacker can form meaningful blocks to prepend or append, since there are only 2^y values to try. If *y* or the bit positions used for the *y* bits vary per block based on the key, an attacker will need to try all values of *y* and possible positions for the *y* bits.

It is not possible to rearrange ciphertext blocks without garbling the plaintext because *y* bits from each plaintext block are used to decrypt the previous plaintext block. In order to swap ciphertext block *i* with ciphertext block *j*, the attacker has to find a ciphertext block in position *i* which, when prepended to the leftmost *y* bits from the $(j+1)^{st}$ plaintext block, will decrypt to a plaintext block whose leftmost *y* bits are the same as the *y* bits appended to the $(j-1)^{st}$ ciphertext block during decryption. Likewise, the j^{th} block must be such that when it is prepended to the leftmost *y* bits from the $(i+1)^{st}$ plaintext block, will decrypt to a plaintext block whose leftmost *y* bits are the same as the *y* bits appended to the $(i-1)^{st}$ ciphertext block during decryption. Furthermore, because the recipient of the ciphertext does not receive the rightmost *y* bits of each block except for the last block. The attacker does not even know all of the ciphertext bits used to decrypt a given block of plaintext when trying to determine what ciphertext blocks can be rearranged without garbling the message.

3.3 Elastic ECB Mode

Our second new mode, shown in Fig. 10, is a possible alternative to ECB mode that offers some protection against pattern detection in and alterations of the ciphertext compared to ECB. In tests, Elastic ECB significantly

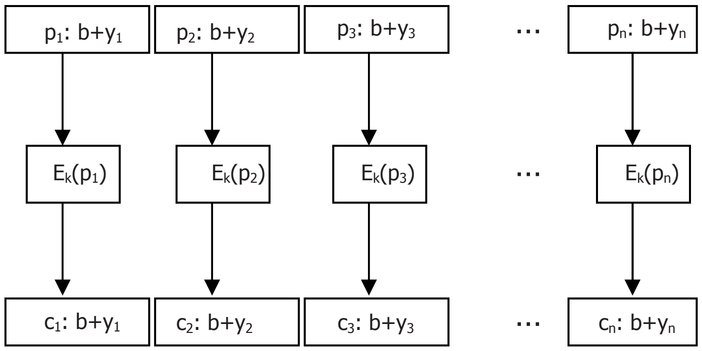


Fig. 10. Elastic ECB mode.

reduced the number of patterns when encrypting data that has repeated plaintext blocks aligning on 16-byte boundaries [4]. The data is encrypted as in ECB mode, but the block size varies per block based on the key, as shown in Fig. 10. The i^{th} block is of length $b + y_i$ for $0 \leq y_i \leq b$ and y_i is based on key bits. The i^{th} block can be decrypted without decrypting any other block by determining its starting position and length from the key. If the key bits are sufficiently random, y_i will be uniformly random within $[0,b]$. Another option is to use key bits to set the first block's length then set each subsequent block size based on bits from the previous ciphertext block, although this will not allow the block lengths to be set in advance.

The i^{th} block can start at any position in the range $b(i-1) + 1$ and $2b(i-1) + 1$, with an average of $(3b(i-1)+2)/2$. For a plaintext pattern to show up in the ciphertext, the starting position of the block (which is now random) and y_i would have to match the starting position of the plaintext pattern and its length in the file. This method does not work for all cases because there are $b+1$ possible block sizes (b to $2b$) if all values of y are used and $(b/8)+1$ possible block sizes if the block size must be an integral number of bytes. If the file is large enough and has a significant number of repeated entries, ciphertext repetitions will occur. The degenerate case is a file consisting entirely of the same byte value repeated, in which case there will be $b/8$ distinct ciphertext blocks if y is restricted to being a multiple of 8.

Replacing individual blocks without garbling the plaintext is possible if the attacker can determine the start and end position of the individual blocks where the modification will occur. Any block being replaced will have to be replaced with a block of the same length; otherwise, the block and all subsequent plaintext blocks will be garbled. The probability of an attacker determining the start and end of the i^{th} block is $1/(b^2(i-1))$. (The probability of guessing the start position of the i^{th} block is $1/(b(i-1))$ and the probability of guessing the length, y_i, of the i^{th} block is $1/b$.) Splicing is even more difficult than replacing individual blocks. If two ciphertexts are being spliced together, the individual block lengths of the result must be the same as the lengths corresponding to the key. If a block is removed, the block boundaries for all subsequent blocks will not correspond to the boundaries used in encryption and the remaining plaintext will be garbled.

Elastic ECB mode is aimed at applications where at least two b-bit blocks are available when encrypting all but the last block so the block size can be varied, with y set to any value in the range of 0 to b. Elastic ECB mode may require a greater amount of computation than ECB due to the need to compute the y_i's from the key and due to varying the block length. Overall encryption time compared to ECB may or may not increase because the longer block lengths will result in fewer blocks to encrypt. The

Table 3. Percent of matching blocks in ECB mode vs elastic ECB mode.

File type	Percent of blocks counting as a match with ECB (10,000 total blocks)	Percent of blocks counting as a match with ECB (max over 10 trials)
Emails	13.62%	0.85%
Email log	34.38%	9.46%
Web log	38.70%	7.49%

total number of rounds required of the elastic ECB mode to encrypt *nb* bits, for some integer $n > 0$, will depend on the n,b and y_i values.

To illustrate how elastic ECB mode reduces patterns, the number of times two or more identical blocks occur within a file was determined when using 16-byte blocks and (16+Y)-byte blocks, where Y is an integer between 0 and 16 that varies per block. We focused the tests on files where patterns are present. English text such as news articles and research papers are unlikely to have repeated phrases that align on 128-bit block boundaries [4]. In contrast, patterns are likely to appear in the ciphertext produced by ECB mode when encrypting structured files where the format of the content results in patterns, such as email logs.

We used files where repetitions of plaintext were frequent but not intentionally aligned on 16-byte boundaries. These files consisted of emails, email logs (SMTP header information) and a log of visitors (IP addresses, and related information) to a web site. The email consisted of emails between three people. The emails were generally short and included forwarded emails but no attached files or images. 160,000 bytes were used from each file. When using elastic ECB mode, the block sizes were determined randomly using the key value as a seed to a random number generator. We ran 10 trials, each with a different key. The results are summarized in Table 3. A block counted as a match if it was identical to any previous block in the file. The maximum number of matches (greatest percent) out of the 10 trials is reported for elastic ECB mode. The number of blocks ranged from 6792 to 6845 across the combined 30 trials of elastic ECB mode on the three file types.

4 Conclusions

The constructions of the elastic versions of AES, Camellia, MISTY1 and RC6 illustrate how to apply the method for creating variable-length block ciphers. By applying the statistical tests used in NIST's AES competition, we conclude that there is no obvious flaw in the design because the level of randomness of the ciphertext produced by each of the elastic versions is consistent with the level required in the AES competition. The workload of the elastic version of a cipher is proportional to the block size, with the number of rounds increasing as the block size increases. The performance benefit from using the elastic version of a block cipher depends on the original cipher and the exact implementation. The percent of overhead involved in adding the swap steps, whitening and two key-dependent permutations varies based on the number of operations and exact implementation of the original cipher. For AES, whose block size is 16 bytes, there is a significant performance benefit when using the elastic version to encrypt blocks up to 25 bytes in length using an implementation of AES that requires little memory; whereas, there is no performance benefit when using a memory intensive implementation that consists entirely of table lookups and XORs. For Camellia, whose block size is 16 bytes, there is a performance benefit when using the elastic version for block sizes up to 19 bytes in length when the initial and final key-dependent permutations are not included. For MISTY1, whose block size is 8 bytes, there is a performance benefit when using the elastic version for block sizes up to 12 bytes. For RC6 with a block size of 16 bytes, there is a performance benefit when using the elastic version for blocks up to 20 bytes in length. The elastic versions offer a performance benefit over previous methods that treat the block cipher as a black box and apply it multiple times.

The ability to encrypt variable-length blocks allows new modes of encryption to be designed. We proposed two ways of using variable-length blocks to create new modes. Elastic Chaining involves processing blocks in a manner such that bits from the i^{th} ciphertext block become part of the $(i+1)^{st}$ plaintext block. When encrypting a sequence of blocks, y bits from the previous ciphertext block are prepended to the current plaintext block to form a $(b+y)$-bit block. This mode prevents the block-wise adaptive attacks to which CBC is subject and, compared to CBC, results in more garbled plaintext blocks when attempting to splice or otherwise alter ciphertext blocks. Elastic ECB mode is ECB mode with key bits determining each block's size such that the block size varies across the blocks. This significantly reduces the probability that patterns are detected, even in highly repetitious data. Furthermore, insertion, removal or rearrangement of blocks requires determining the start position and length of the blocks. These proposals for modes of encryption are intended as initial concepts to demonstrate additional potential uses of elastic block ciphers and require further analysis.

Acknowledgments

This work was partially supported by NSF Grants ITR CNS-04-26623 and CPA CCF-05-41093. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the NSF or the U.S Government.

References

[1] K. Aoki, T. Ichikawa, M. Kanda, M. Matsui, S. Moriai, J. Nakajima, and T. Tokita. “Camellia: A 128-Bit Block Cipher Suitable for Multiple Platforms – Design and Analysis”. In Proceedings of Selected Areas in Cryptography, LNCS 2012, Springer-Verlag, pp. 39–56, 2000.
[2] M. Bellare and P. Rogaway. “On the Construction of Variable Length-Input Ciphers”. In Proceedings of Fast Software Encryption, LNCS 1636, Springer-Verlag, 1999.
[3] M. Ciet, G. Piret, and J. Quisquater. “Related-Key and Slide Attacks: Analysis, Connections and Improvements, Extended Abstract”. UCL Crypto Group Technical Report, 2002.
[4] D. Cook. “Elastic Block Ciphers”. Ph.D. Thesis, Columbia University, New York, 2006.
[5] D. Cook, M. Yung, and A. Keromytis. “Elastic Block Ciphers: The Basic Design”. In Proceedings of ASIACCS, ACM, pp. 350–355, March 2007.
[6] J. Daemon and V. Rijmen, “The Design of Rijndael: AES the Advanced Encryption Standard”. Springer-Verlag, Berlin, 2002.
[7] A. Joux, G. Martinet, and F.Valette. “Blockwise-Adaptive Attackers: Revisiting the (In)Security of Some Provably Secure Encryption Models”. In Proceedings of Advances in Cryptology – CRYPTO, LNCS 2442, Springer-Verlag, August 2002.
[8] M. Matsui, “New Block Encryption Algorithm MISTY”. In Proceedings of Fast Software Encryption, LNCS 1267, Springer-Verlag, pp. 54–68, 1997.
[9] I. Mironov. “(Not So) Random Shuffles of RC4”. In Proceedings of Advances in Cryptology – CRYPTO, LNCS 2442, Springer-Verlag, August 2002.
[10] “NESSIE Security Report, Version 2”. https://www.cosic.esat.kuleuven.ac.be/nessie, February 2003.
[11] NIST. “Randomness Testing of the Advanced Encryption Standard Finalist Candidates”, http://citeseer.ist.psu.edu/soto00randomness.html, March 2000.
[12] NIST. “A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications”. NIST Special Publication 800–22. http://www.csrc.nist.gov/publications/nistir, 2001.
[13] NIST. “FIPS 197 Advanced Encryption Standard (AES)”, http://www.csrc.nist.gov/publications/fips/ fips197/fips-197.pdf, 2001.

[14] NIST. “FIPS 180-2 Secure Hash Standard”, http://www.csrc.nist.gov/publications/fips/fips180-2/fips180-2withchangenotice.pdf, 2002.
[15] S. Patel, Z. Ramzan, and G. Sundaram. “Efficient Constructions of Variable-Input-Length Block Ciphers”. In Proceedings of Selected Areas in Cryptography, LNCS 3357, Springer-Verlag, 2004.
[16] random.org. http://wwww.random.org/files.
[17] R. Rivest. “RC4”. In Applied Cryptography by B. Schneier, John Wiley and Sons, New York, 1996.
[18] R. Rivest, M.J.B. Robshaw, R. Sidney, and Y.L. Yin. “RC6 Block Cipher”. http://www.rsa.security.com/rsalabs/rc6, 1998.

6
Vulnerability Response Decision Assistance

Hal Burch[1], Art Manion[1], and Yurie Ito[2]

[1] Carnegie Mellon University, Pittsburgh, PA, USA
{hburch, amanion}@cert.org
[2] Japan Computer Emergency Response Team Coordination Center, Tokyo, Japan
yito@jpcert.or.jp

Abstract. Each year, thousands of new software vulnerabilities are reported, and affected organizations must analyze them and decide how to respond. Many organizations employ ad hoc systems of decision making, which often result in inconsistent decisions that do not properly reflect the concerns of the organization at large. VRDA (Vulnerability Response Decision Assistance) allows organizations to leverage the analysis effort at other organizations and to structure decision-making. VRDA enables organizations to spend less time analyzing vulnerabilities in which they are not interested, to make decisions more consistently, and to structure their decision making to better align with the goals of the organization. VRDA consists of a data exchange format, a decision making model, a decision model creation technique, and a tool embodying these concepts. One response team is employing a basic form of VRDA to cut the number of vulnerabilities analyzed by a factor of two. Another response team is developing and testing a VRDA implementation within their organization.

1 Introduction

In 2006, CERT/CC recorded more than 8,064 vulnerabilities [1] and NVD recorded 6,604 vulnerabilities [2]. For each vulnerability, organizations must analyze the vulnerability to determine which software systems are affected, the impact of a successful exploit, and how difficult it is for an attacker to successfully exploit the vulnerability. Once this analysis is

V. Siris et al. (eds.), *Proceedings of the 3rd European Conference on Computer Network Defense*, Lecture Notes in Electrical Engineering,
DOI 10.1007/978-0-387-85555-4_6, © Springer Science+Business Media, LLC 2009

complete, the organization must determine whether or not the vulnerability warrants further action, whether that be producing an alert, conducting further analysis, or otherwise responding to the vulnerability. This expensive analysis is repeated at organizations around the world, resulting in a large duplication of effort.

Once the analysis is complete, an organization must decide how they will respond to the vulnerability. Responses vary; it may ignore the issue, immediately starting the patching process, record the issue for the next periodic update, or publish the information about the vulnerabilities (internally or externally). The actions warranted by a particular vulnerability are based on the number of systems affected, the value of those systems, how likely exploit is perceived to be, the impact of a successful exploit, and the risk and expense induced by testing the patch and deploying it.

In many organizations, the decision about which actions are warranted is made in an ad hoc manner based on staff experience. This results in incorrect and inconsistent decisions that tend to reflect the goals of the organization as perceived by the decision maker, not the true organizational goals. Multiple decision makers only confound the problem.

We propose a new system called VRDA (Vulnerability Response Decision Assistance). VRDA address these problems by giving structure to the decision making process. This structuring enables interchange of the analysis, reducing the duplication of effort. In addition, the concerns of the organization can be codified into a model, making the decisions more consistent and better aligned with the organizational goals. VRDA is designed to answer the questions about which vulnerabilities an organization should be responding to, what the response should be, and with what priority.

2 VRDA

VRDA is composed of multiple components: a break-down of facts (attributes or properties of vulnerabilities), a method for recording relationship between affected systems and fact values, a data exchange architecture, a data exchange format, a format for modeling decision, and a method for creating decision models. To maximize the value that organizations can derive from it, VRDA is designed to be open and modular. Thus, an organization need only use the components that best meet their needs. The various components of VRDA are explained throughout this paper. An overview of VRDA usage is shown in Fig. 1.

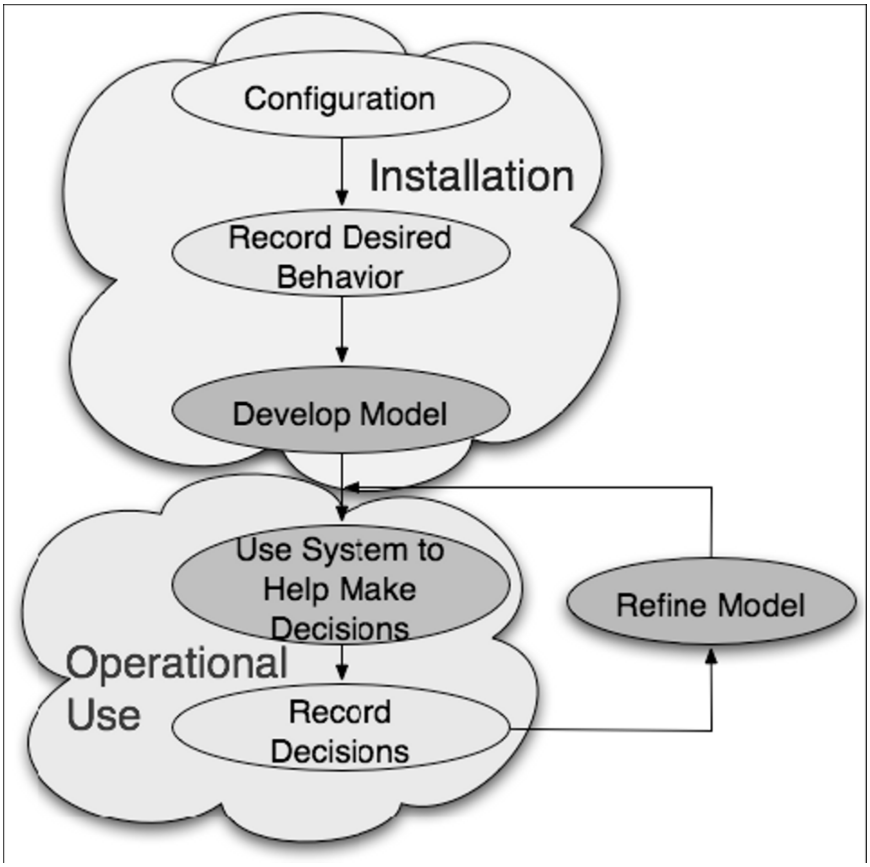


Fig. 1. VRDA system overview.

During the installation phase, the organization using VRDA must determine what information (facts) is necessary to make accurate response decisions (perform tasks). VRDA is configured with the selected facts and tasks. Facts may be obtained from an upstream provider; in otherwords, the organization may subscribe to a feed of VRDA facts from a computer security incident response team (CSIRT) that has vulnerability analysis capability. Analysts knowledgeable about the organization's IT assets and business operations must then train the system, recording the appropriate response (desired behavior) for a set of sample vulnerability reports. This process encodes the organization's values into a decision model.

With a working decision model, VRDA can enter the operational phase. As analysts score vulnerabilities, or append organization-specific data to the upstream feed, VRDA suggests appropriate responses. The actual response decision is also recorded, so that the model can be refined if necessary. For example, if VRDA regularly suggests a certain response, but analysts often disagree, then the model most likely does not accurately reflect the organization's values, or VRDA may not be configured with the necessary facts.

VRDA is designed to tolerate incomplete information and "best guesses" by analysts. VRDA often does not examine every fact to reach a decision, and even a few off-by-one scoring errors are unlikely to significantly affect the decision. A well-constructed model will quickly and accurately enable easy decisions, such as ignoring vulnerability reports that do not affect the organization or flagging high severity reports.

2.1 Facts

In VRDA terms, *facts* are assertions about vulnerabilities. Facts can also be thought of as attributes, characteristics, or properties of vulnerabilities. The act of an analyst researching and recording facts is called *scoring*. A VRDA fact is not a "fact" in the strict definition, that is, VRDA facts are not indisputable and may be subject to interpretation. Facts represent the best information available to the analyst at the time of scoring, and fact values may be determined in part by analyst judgment and experience.

VRDA proposes a set of "core" facts. An organization may, however, create and score any additional facts that affect response decisions. A VRDA instance that does not consider significant facts will likely produce incorrect suggestions. In other words, the set of facts used by a VRDA instance is not fixed.

Most facts have an ordered range of possible values. To balance the value of accuracy with the cost of achieving accuracy, the granularity for fact values is fairly low. For example, a common range contains four possible values: "low," "low-medium," "medium-high," and "high." VRDA prefers four values to reduce the tendency of analysts to select the median "safe" value. When lacking sufficient information to make a confident selection, analysts should make an intuitive, educated guess.

VRDA facts are organized into three categories: vulnerability facts, world facts, and constituency facts. The distinctions between categories do not affect the decision modeling. The distinctions do help inform how the facts apply and whom should provide them.

Vulnerability facts apply directly to vulnerabilities. These facts are typically inherent technical properties of vulnerabilities. Vulnerability facts apply regardless of other world or constituency facts. For example, a denial-of-service impact exists whether or not exploit code is available or the vulnerable software is deployed within a given constituency. While anyone can score vulnerability facts, VRDA expects an experienced analyst (e.g., a CSIRT or response team) to score vulnerability facts and provide them to downstream consumers. The following is a working list of vulnerability facts. Further explanation is provided in VRDA documentation.

Security Product – Does the vulnerability affect a security product? (Yes/No)

Network Infrastructure Product – Does the vulnerability affect a network infrastructure product? (Yes/No)

Multiple Vendors – Does the vulnerability affect multiple vendors? (Yes/No)

Impact 1 – What is the general level of impact of the vulnerability on a system? (Low, Low-Medium, Medium-High, High)

Impact 2 – What are the levels of impact for confidentiality, integrity, and availability of the vulnerability on a system? (Low, Low-Medium, Medium-High, High)

Access Required – What access is required by an attacker to be able to exploit the vulnerability? (Routed, Non-routed, Local, Physical)

Authentication – What level of authentication is required by an attacker to be able to exploit the vulnerability? (None, Limited, Standard, Privileged)

Actions Required – What actions by non-attackers are required for an attacker to exploit the vulnerability? (None, Simple, Complex)

Technical Difficulty – What degree of technical difficulty does an attacker face in order to exploit the vulnerability? (Low, Low-Medium, Medium-High, High)

World facts apply to "meta" information about vulnerabilities. World facts describe states of the world or environment in which vulnerabilities exist. World facts apply to all constituencies. Anyone can score world facts, although VRDA expects experienced analysts to score and provide them to downstream consumers. The following is a working list of world facts. Further explanation is provided in VRDA documentation.

Public Attention – What amount of public attention is the vulnerability receiving? (None, Low, Low-Medium, Medium-High, High)

Quality of Public Information – What is the quality of public information available about the vulnerability? (Unacceptable, Acceptable, High)

Exploit Activity – What level of exploit or attack activity exists? (None, Exploit exists, Low activity, High activity).

Report Source – What person or group reported the vulnerability?

Constituency facts measure information about vulnerabilities that is specific to a given constituency. For example, a CSIRT may consider a large scope, like entire site, or even the entire Internet, when deciding how to respond. A system administrator may consider a smaller scope, such as a site, lab, or business unit. Constituency facts most likely differ between constituencies and must be provided by (or on behalf of) the constituency making the response decision.

Population – What is the population of vulnerable systems within the constituency? (None, Low, Low-Medium, Medium-High, High)

Population Importance – How important are the vulnerable systems within the constituency? (Low, Low-Medium, Medium-High, High)

Default Fact Sets (DFS) are preset fact values for a set of facts. DFS help analysts consistently score similar vulnerabilities. DFS was conceived to help score impact. For example, an analyst may determine that most vulnerabilities that allow an attacker to execute arbitrary code have the value "high" for the confidentiality, integrity, and availability *Impact 2* facts. The analyst could define a DFS named "execute arbitrary code" that sets the *Impact 2* facts when the DFS is applied to a vulnerability report. In addition to applying the default fact sets, the name of the DFS itself is a fact (i.e., that the analyst applied the DFS named "execute arbitrary code" may factor into the response decision). In this sense, a DFS is similar to a keyword or tag.

2.2 Light-Weight Affected Product Tags

One barrier to the exchange of facts about vulnerabilities is that some important facts, such as population size and population importance, are constituency facts that cannot be determined by an outside entity. If the population of a particular affected system is high in one constituency, it does not guarantee the population is high in any subset of that constituency. For example, Microsoft Internet Explorer has a high population on the internet as a whole, but a given organization may have few systems using Microsoft Internet Explorer, either because they, by policy, use a different browser or because they have few Microsoft Windows systems.

As a result, constituency facts are generically not helpful to communicate. Light-weight affected product tags (LAPTs) communicate the set of affected systems so that the constituency facts can be computed. A vulnerability is tagged with the list of affected systems encoded as LAPTs. When VRDA receives a vulnerability with LAPTs, it consults a database to determine the constituency fact values for each LAPT listed and, for each constituency fact, selects the value corresponding to the worst (most likely to result in action) value appearing in the LAPTs. For example, if a vulnerability affects a high population LAPT and a low population LAPT, the population is set to high. The worst value is selected as the best guess and the least likely to hide an important vulnerability. Few vulnerabilities affect a large set of low population systems that together represent a high population.

This combination can lead to perhaps unexpected results, when a low population and high value product is affected along with a high population but low value product. The resulting constituency facts are a high population size and value, which is arguably incorrect. The root of the problem, however, is not LAPTs but rather simplification in the fact modeling. A population with a few high-value systems and a lot of low-value systems is difficult to model in terms of purely population size and population value. We believe that modeling as we have is more useful, but if an organization was so inclined, VRDA can be configured to population sizes for each population value, in which case this problem would not arise.

LAPTs are designed to be general descriptions of the affected system. Thus, it specifies the vendor and the system. Unless a population is large and many vulnerabilities affect only particular versions, the version is not included. For example, the Apache web server is a single LAPT. The motivations for this lack of specificity are as follows:

1. Difficulty of determining version information: It is often difficult to determine the exact versions affected, particular for commercial systems or systems that don't require patches to be applied in a particular order.
2. Cost of inventory maintenance: It is difficult enough to maintain inventory of the size of each software product. Keeping track of version numbers can make the problems an order of magnitude worse.
3. Limited usefulness: Most vulnerabilities discovered affect the current version of the affected software product, and often all recent versions. Thus, few vulnerabilities will be discarded by checking version numbers.

Most LAPTs refer to particular software products. When a vulnerability affects a technology or is an error common to many implementations of a technology, a "technology LAPT" can be utilized. For example, an error that is observed in SSL or in many SSL libraries might be attributed to the technology of "SSL", rather than listing all the affected products, although the list of LAPTs might also include products known to be affected. For common technologies, any list of products using the technology is doomed to be incomplete. Technology LAPTs provide a mechanism to describe a class of affected products to avoid this incompleteness.

2.3 Data Exchange

One of the goals of VRDA is to reduce the redundant analysis of vulnerabilities that takes place today. To this end, an organization must be able to obtain structured vulnerability information, in the form of fact values and LAPTs from another organization. Once an organization receives this information, they may refine the fact values or augment with additional fact values as desired and republish the results for use by other organizations. For example, a CSIRT with national responsibility might publish the information for a company's CSIRT who then passes information on to groups internal to that company. This second level might be in the form of alerts or a subset of the vulnerabilities in VRDA format, perhaps augmented with additional local information.

When an organization receives vulnerability information in the form of fact values and LAPTs, any previously unknown LAPTs must have their values recorded. Once they are recorded or if there are no new LAPTs, the constituency facts can be added to the vulnerability information. At this point, VRDA filters out based in its configuration. Ideally, VRDA would filter out the majority of the vulnerabilities, enabling an organization to focus its resources better. For vulnerabilities that pass this filter, the organization adds fact values for any local facts. Then, decisions are suggested based on the process described in "decision modeling."

The data exchange format is based on VULDEF [3]. Most of the optional fields of VULDEF, such as Solution, Related, and Exploit, are ignored by VRDA. The major differences from VULDEF to the VRDA exchange format is AffectedItem is augmented with a LAPT and FactList and facts tags are added to communicate the values of the facts.

An abbreviated example of the interchange format

```
<?xml version="1.0" encoding="UTF-8"?>
<Vulinfo>
  <VulinfoID>JVN#178394</VulinfoID>
  <VulinfoData>
    <Affected version="1.0" historyno="2">
      <AffectedItem lapt="Microsoft-Windows-XP"/>
      <AffectedItem lapt="Tech-HTTP-Server"/>
    </Affected>
    <FactList version="1.0" historyno="2">
      <ImpactConfidential-
ity>Low</ImpactConfidentiality>
      <ImpactAvailability>Low</ImpactAvailability>
      <ImpactIntegrity>Medium</ImpactIntegrity>
      <AccessRequired>Routed</AccessRequired>
...
</VulinfoData>
</Vulinfo>
```

2.4 Decision Modeling

VRDA makes decisions by modeling the process as a decision tree [4]. An example of a decision tree is shown in Fig. 2. The evaluation of a decision tree begins at the root. At each node along the evaluation path, the child based on the attribute associated with that node. In the example decision tree, if the population is high, then the left child is followed, at which point the difficulty of exploit is considered. If the difficulty of exploit is low, then the decision tree evaluates to “must”.

We selected decision trees because their behavior is clearly indicated and they can be hand-modified. Although we expect decision trees to be computed based on recorded decisions for past vulnerabilities, the computed trees might need refinement. For example, an organization may, as policy, not need independently verify reports from particular sources or may always respond to particular types of reports. Modifying a decision tree to represent such policies can be done clearly and simply. Alternative decision models, such as neural nets or linear combinations, might be able to better capture the intricacies of the decision making process, but it is difficult to understand what policy they implement and they lack the ability to predictably hand-tune the model.

We expect the resulting decisions to be imperfect. However, we compensate for that by giving gradients of decisions, rather than Boolean values. In particular, we use four levels: “must”, “should”, “might”, and “won’t”. The goal is that the resulting decision level should not differ more than one from the “correct” value. Since, in our experience, experts often disagree more widely than that anyway, this accuracy may be ambitious. In any case, the decisions serve as a guideline, rather than a rule, to handlers. This allows for automated prioritization, including deciding to ignore

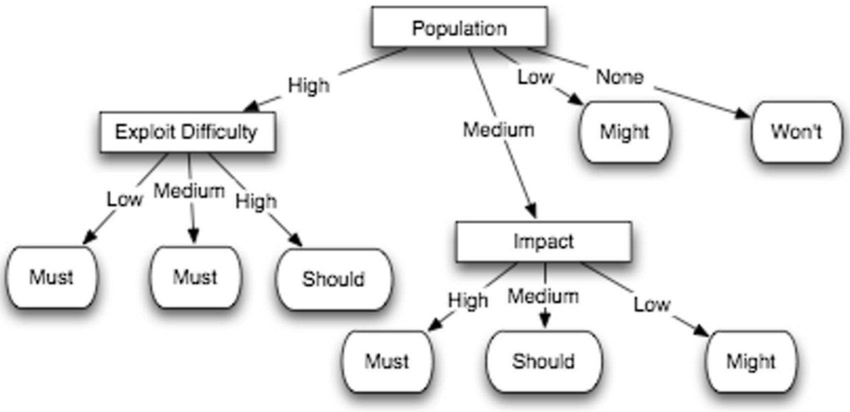


Fig. 2. An example of a decision tree.

vulnerabilities whose evaluation falls says "won't" for all decisions, which reduces the load on handlers within an organization VRDA constructs decision trees using a similar algorithm to the one described in two references ([4] and [5]) which is based on a recursive selection of the attribute that reduces entropy the most. However, instead of pruning the decision tree after its construction, attributes that fail the chi-squared test of significance are not considered when constructing the tree.

Once the decision tree is constructor, the organization can modify the decision tree, change leaf values, changing the attribute used at a node, and have the algorithm compute a leaf value or an entire sub-tree. These operations allow the organization to codify an important and well-understood policy decision, such as always patching certain types of vulnerabilities manually and then allowing the tree for the other vulnerabilities to be computed automatically.

3 Current Usage

CERT/CC uses a limited implementation of VRDA to decide which vulnerability reports require the attention of a human analyst. A brief study showed that two facts heavily influenced this decision: the population of affected systems and the broad impact of the vulnerability. By recording these two facts and applying a simple decision model, the number of reports assigned to analysts was reduced by half.

JPCERT/CC has developed a web-based VRDA implementation called KENGINE. JPCERT/CC is using KENGINE internally and is testing with several constituents. In addition to implementing the VRDA specification, KENGINE provides workflow management and reporting features to monitor performance and decision behavior. JPCERT/CC is planning to publish vulnerability information in VRDA format and to release KENGINE to the public.

4 Future Direction

The teams developing VRDA expect to do the following:

1. Score vulnerability reports and provide facts to downstream consumers, possibly the general public. Consumers may use the facts as they wish, with or without the decision support component.
2. Provide documentation and guidance for consumers to use the decision support component.

3. Deploy beta VRDA systems to knowledgeable consumers to gain real-world experience and determine if VRDA is useful and viable.
4. Refine VRDA according to results of the beta deployments and community feedback.
5. Examine interoperability with other vulnerability information systems, particularly the Common Vulnerability Scoring System (CVSS) [6].

5 Related Work

A number of efforts have been made (or are currently underway) to represent vulnerability information and provide consumers some means to determine severity, leading to an appropriate response. These efforts include metrics, information exchange formats and protocols, and vulnerability information databases. VRDA draws ideas and inspiration from many of these efforts while proposing a decision support approach to vulnerability response.

5.1 Common Vulnerability Scoring System (CVSS)

Perhaps the most similar and contemporary effort, the Common Vulnerability Scoring System (CVSS) "… is designed to rank information system vulnerabilities and provide the end user with a composite score representing the overall severity and risk the vulnerability presents." [6] VRDA is similar to CVSS in some ways, particularly in the representation of facts (VRDA) and metrics (CVSS) used to describe vulnerabilities. It may well be possible to use the decision support component of VRDA with CVSS metrics. While VRDA specifies a set of core facts, any fact that contributes significantly to a response decision for an organization can and should be considered. In contrast, CVSS specifies a fixed list of metrics.

A more notable difference is that VRDA uses decision support concepts to generate individual response decisions, while CVSS assigns fixed values to metrics and applies a single equation to calculate severity. VRDA effectively allows organizations to set their own individual values and make their own individual response decision. This design choice comes at a cost – VRDA requires more effort on the part of the organization than CVSS. And in fairness, CVSS does provide limited environmental metrics that modify the overall score based on characteristics that are unique to individual organizations.

Although CVSS and VRDA measure vulnerability characteristics similarly, the two systems are designed with somewhat different goals. CVSS aims to provide an overall severity score, while VRDA focuses on the decision-making aspect of how an individual organization responds to vulnerabilities.

5.2 Exchange Formats

There exist a variety of vulnerability information exchange formats, including the Common Announcement Interchange Format (CAIF) [7], the Common Model of System Information (CMSI) [8], the EISPP Common Advisory Format Description [9] and the Deutsches Advisory Format (DAF) [10]. These formats generally describe ways to exchange and present vulnerability information for use in advisory documents and include functionality that is unnecessary for VRDA. In some cases, the formats cannot be reasonably extended to meet VRDA requirements.

5.3 Other Work

Other related work includes (in no particular order): the Purdue University CERIAS Cassandra tool [11], the CERT Coordination Center/US-CERT Metric [12], MITRE Common Vulnerabilities and Exposures (CVE) [13] and Open Vulnerability and Assessment Language (OVAL) [14], the VULnerability Data publication and Exchange Format (VULDEF) [3] (used as the basis for the VRDA exchange format), NIST ICAT (now deprecated) [15], the National Vulnerability Database (NVD, successor to ICAT) [2], the, the Vulnerability and eXposure Markup Language (VuXML) [16], the Open Source Vulnerability Database (OSVDB) [17], and SIGVI [18].

References

[1] CERT/CC Statistics 1988 – 2006, http://www.cert.org/stats/

[2] National Vulnerability Database (NVD) Statistics, http://nvd.nist.gov/statistics.cfm

[3] Terada, M.: VULDEF: The VULnerability Data publication and Exchange Format data model, http://jvnrss.ise.chuo-u.ac.jp/jtg/vuldef/index.en.html

[4] Russell, S., Norvig P.: Artificial Intelligence: A Modern Approach. Prentice-Hall, Englewood Cliff, NJ (1995)

[5] Moore, A.: Decision Trees, http://www.autonlab.org/tutorials/dtree.html

[6] Forum of Incident Response Teams: Common Vulnerability Scoring System (CVSS), http://www.first.org/cvss/, http://www.first.org/cvss/cvss-guide.html

[7] RUS-CERT: Common Announcement Interchange Format (CAIF), http://www.caif.info/

[8] Grobauer, B.: CVE, CME,..., CMSI? – Standardizing System Information, http://www.first.org/conference/2005/papers/dr.-bernd-grobauer-paper-1.pdf

[9] European Information Security Promotion Programme (EISPP): Common Advisory Format Description 2.0, http://www.eispp.org/commonformat_2_0.pdf
[10] Deutscher CERT-Verbund: Deutsches Advisory Format (DAF), http://www.cert-verbund.de/daf/index.html, 2004.
[11] CERIAS Cassandra tool, https://cassandra.cerias.purdue.edu/main/index.html
[12] US-CERT Vulnerability Notes Field Descriptions – Metric, http://www.kb.cert.org/vuls/html/fieldhelp#metric
[13] Common Vulnerabilities and Exposures (CVE), http://cve.mitre.org/
[14] Vulnerability and Assessment Language (OVAL), http://oval.mitre.org/
[15] ICAT Metabase, http://icat.nist.gov/icat_documentation.htm, http://web.archive.org/web/20050320143644/http://icat.nist.gov/icat_documentation.htm
[16] Vulnerability and eXposure Markup Language (VuXML), http://www.vuxml.org/
[17] OSVDB: The Open Source Vulnerability Database, http://osvdb.org/
[18] SIGVI, http://sigvi.sourceforge.net/what_is.php

7

Alice, What Did You Do Last Time? Fighting Phishing Using Past Activity Tests

Nikos Nikiforakis, Andreas Makridakis, Elias Athanasopoulos, and Evangelos P. Markatos

Institute of Computer Science, FORTH, Heraklion, Greece
{nikifor, amakrid, elathan, markatos}@ics.forth.gr

Abstract. Phishing attacks are one of the most crucial modern security threats in the current World Wide Web. An adversary may clone a legitimate Web site and lure a user to submit her credentials to the malicious construct. The adversary may then use the stolen credentials to the authentic site. In this paper we present a novel idea to fight phishing using Past Activity Tests (PACTs). In a nutshell, PACTs take advantage of the fact that the user has accessed at least once her account in the past, contrary to the phisher who accesses the user's account for the first time. Thus, a user can answer a question relative to her past activity, but the attacker can not.

1 Introduction

Phishing attacks exploit social engineering techniques in order to lure users and convince them to give their account information, such as their login and their password, in a fraud web site. A phishing attack is held by building a web site identical of the authentic one. This process does not require cloning the legitimate web site in whole, but only the pages involved in the login process of a user, in order to give the impression to the victim that she is visiting the authentic web site and not the cloned one.

Phishing attacks are considered as a major security threat in modern Internet. In Phishing Attack Trends Report, published by Anti-Phishing

Working Group (APWG) [1] in April 2007, it was reported that the number of unique phishing web sites detected by APWG rose to 55,643 in April 2007. A massive jump of nearly 35,000 from March was detected, resulting from aggressive sub-domain phishing tactics by which phishers started using the tactic of putting a large number of phishing URLs on the same domain. A large number of banks were targeted in April with seven of the most-targeted 20 brands in that month belonging to European banks. In addition, one of the top 20 was a Canadian financial institution.

Phishing is usually connected with the cloning of sites that are active in e-commerce, like bank services or on-line stores. However, a phishing attack may target sites that are not involved directly with money transactions. These sites may still embed private social information which is sensitive and must not be leaked to third parties. Some examples are, blog services, e-mail services or content hosting services, like Flickr.com.[1] These sites are expected to be phishing targets with a lower probability than e-commerce sites, which may not be willing to invest a large amount of money for an anti-phishing approach that requires an extensive effort and modifications to the site's infrastructure.

In this paper, we present a novel anti-phishing approach, which:

- is easily deployable in the server side of a web site,
- does not need client-side modifications to a web browser,
- is low cost,
- is user friendly.

Our anti-phishing solution adds an extra authenticating step between the username/password form and the complete authentication of the user. It is based on the fact that *the phisher is accessing the victim's account for the first time in contrast with the legitimate user, who has accessed her account in the past*. The rest of this paper is organized as follows. In Section 2 we present our anti-phishing approach by introducing for the first time Past Activity Tests (PACTs). In Section 3 we evaluate theoretically the PACT approach and we highlight PACTs' limitations. In Section 4 we present two real deployments. We present related work in Section 5 and we conclude in Section 6.

[1] A popular site, which hosts a user's pictures in private or public areas.

2 PACT Architecture

This section highlights the basic components of an anti-phishing architecture, which is based on Past Activity Tests (PACTs). We assume Alice is a registered user to an on-line service and Trudy an adversary, who tries to steal Alice's credentials using phishing.

2.1 PACT Definition

We define a PACT as follows:

PACT definition: *A PACT is a dynamic test with N possible answers but only one solution. The solution is correlated with Alice's past activity of her account. The correct solution can only be given by the user with probability 1, or by Trudy with probability 1/N.*

Following directly from PACT definition, the idea we are building over our anti-phishing solution, relies on the following observation:

Alice has visited her account at least once in the past. On the contrast, Trudy has access in Alice's active account for the first time.

Using PACT we ask Alice to give an answer to an obvious question, which is based on her past activity to her account. This answer can only be guessed by Trudy, since she is not the real owner of the account and thus, has no knowledge of its past activity.

2.2 Example PACTs

In order to deploy PACT in a real on-line service, the service must create dynamic puzzles based on its subscribers profiles. These puzzles must be solved upon a user is authenticating herself to the on-line service. In order to build a PACT a service must have the following information:

- A summary of Alice's past actions.
- A pool of abstract actions to the service performed by a speculative subscriber.

For example in an e-mail service, Alice may be asked to choose from a pool of e-mail addresses, an e-mail address with which she had contact in the near past. In Table 1 we list a series of PACT examples.

Table 1. Example of PACTs for different possible on-line services.

Service	PACT
E-mail	Select an e-mail address you had contact with in the near past.
E-commerce	Select an item you have purchased in the past.
Content host	Select a picture you have uploaded in the near past[2].
Instant messaging	Select a user, who is in your contact list[3].

3 PACT Evaluation

In this section we present a theoretical evaluation of PACTs and highlight their limitations.

3.1 PACT Resistance

Following directly from PACT definition, the probability of solving a PACT is $p_s = 1/N$, where N denotes the number of possible answers. An interesting property of PACTs is that they can be combined so as the probability p_s to degrade exponentially. For example, by combining m PACTs we have $p_s = 1/ N^m$. Assuming that the on-line server suspends an account after a false attempt, a brute force approach for by-passing a PACT is considered unrealistic. Between the exploiting of an on-line account, by malicious users, and the temporary suspension of it, the latter is preferable.

3.2 PACT Suspension Policy

PACTs can adapt their suspension policy, according to the type of service provided and the available amount of data from the user.

In an e-mail service a user has probably sent more than one e-mails in the near past. So, if she fails to answer the first PACT puzzle, she is provided a new puzzle with a different correct answer. Thus, the account will be suspended after two unsuccessful attempts.

On the contrast, in a bidding service a user will more likely find the correct answer to the first PACT puzzle, since the question concerns an item bought by her in the past. Thus, the account can be suspended after the first unsuccessful attempt.

[2] The majority of popular content host providers allow you to maintain a private area with your content. We assume that PACTs solution is taken from Alice's private area.

[3] This is a case where PACT is used to an on-line service that is not necessarily-hosted in the World Wide Web.

3.3 PACT Limitations

PACT can resist to a phishing attack since the probability for an adversary for solving a PACT is quite low. However, PACT can not resist to a Man-in-the-Middle attack. If Trudy can set up a Phishing Proxy between Alice and the on-line service, then Trudy, upon reaching the PACT, can redirect the test to Alice and get the solution.

Although PACT can not resist to a MiM attack, it actually prohibits the attacker of creating a collection of stolen accounts that she can later use and/or distribute. Even if an attacker manages to authenticate a session of a user, she can only use it at that time, until the session expires (e.g. bank sessions expire within hours). When the session expires, a new PACT puzzle will be introduced and the attacker will not be able to solve it. So mass-phishing is doable but cumbersome.

Fighting the Phishing Proxy threat model is beyond the scope of the PACT solution. However, there are other countermeasures for malicious Proxies. The most widely adopted is the use of Cryptographic Certificates. It is assumed, that the malicious Proxy can not have access to the authentic server's certificate. Also, it is not easy for an attacker to register a security certificate that obviously tries to impersonate a valid well-known company (e.g. citybank vs citybanc). Then, it is a matter of the adversary to convince the user to accept the forged certificate.

We also acknowledge that an implementation of PACTs in an e-mail service may pose a minor privacy issue, because one out of N presented e-mail addresses will be a valid contact. But without PACTs the attacker already has full access to the victim's mailbox. So, though we may not completely solve the privacy problem, we are increasing the overall security of the service provided.

4 Case Studies

In this section we present two real deployments of on-line services, which use PACT as an anti-phishing countermeasure.

4.1 A PACT Enabled E-Mail Service

This case study refers to a secure way of management an e-mail account. Our goal is to prevent Trudy from accessing Alice's e-mail account. When Alice authenticates herself to the e-mail service, then the service requires from her to answer the following PACT: "Please choose an e-mail address, in which you have sent an e-mail in the near past". A list of ten e-mail addresses is presented and Alice must choose the valid one.

In the left frame of Fig. 1 we present a screenshot of the e-mail service, which has been developed using PHP [2]. In order to store the information needed for the dynamic creation of PACTs, we used a PostgreSQL DataBase [3] (version 8.1.8).

Our implementation scheme consists of three tables. The first one, named "*users*", has four fields: *userid, username, password* and *suspended*. Every tuple of this table correspond to a specific e-mail user. The second table, named "*emails*", has three fields: *id*, *address* and *valid*. The tuples of this table represent valid and invalid e-mail addresses. The last table, named "*emailsFromUsers*", has two fields: *user_id* and *e-mail*. It contains both valid and invalid (fake) e-mails which will be used by PACT.

In order to test the service we collected 900 e-mail addresses from Yahoo public lists [4], using a script written in Python. 85.3% of these addresses correspond to yahoo domain. For the purposes of our experiment, we assumed that 800 of the above e-mails are invalid and the rest 100 are valid. All these e-mail addresses were inserted, as tuples, in table "emails". The "valid" field is false for the 800 tuples, which represent invalid e-mails and true for the rest 100 valid e-mail addresses.

Alternatively, construction of invalid e-mail addresses can be used instead of harvesting them off the Internet. According to the results of a quick experiment, 8 out of 10 real e-mail addresses do not produce results in Google [5]. So, a malicious user can not easily find out the correct answer of a PACT puzzle in time.

In the initial page of the e-mail service, Alice must insert into the login form the correct username and password. In case of valid input, a list of ten e-mail addresses is presented. Nine addresses are randomly selected tuples from table e-mails, where field "valid" is false, and the other address is randomly selected from the same table, where field "valid" is true. If Alice selects the valid e-mail address, a new list of ten addresses will be presented. The new list is created with the same mechanism, as mentioned before. Each time, the ids of these 10 addresses, along with the proper user id, are inserted in table "emailsFromUsers". All the previous tuples of this table, that correspond to the user trying to login, are deleted. Since Alice must choose twice a valid e-mail from a list of ten e-mails, the probability Trudy to efficiently log in Alice's e-mail account is *(1/10) * (1/10)*, namely *1/100.*

In the case where Alice inserts into the login form her valid username and password, her account is temporally suspended. This is achieved via the assignment of value "true" in the field "suspended" of table "users", in the tuple that corresponds to Alice. Thus, Trudy can not observe some of the puzzles in order to infer the list of valid e-mail addresses.

Alice's account becomes unsuspended, exclusively if she answers correct both questions. However there are possible situations where Alice:

1. fails to answer one of the two questions, selecting an invalid e-mail address
2. answers any of the two questions, but in a time interval longer than 60 seconds
3. never answers the puzzle
4. visits another webpage and then forwards again on the webpage of the questions.

In all the above cases, Alice's account remains suspended and she must contact the site's administrator, in order to unsuspend her account.

For administration purposes, the e-mail service keeps log files, where information about visiting users is appended. Particularly, information is appended in the following cases:

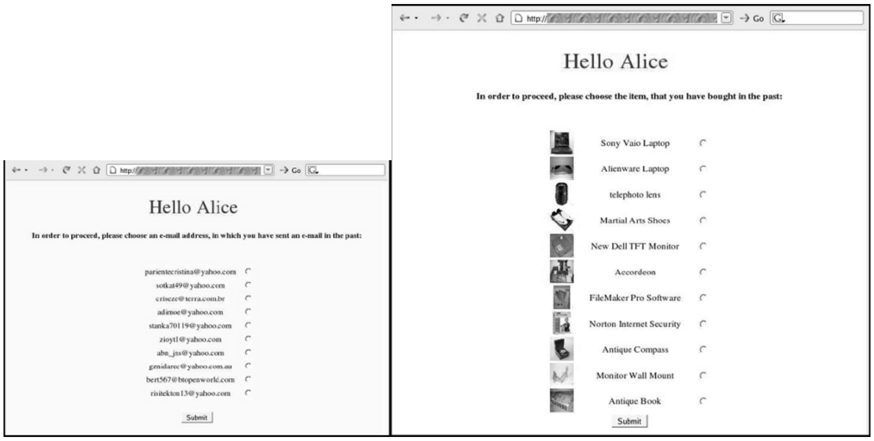
- the login form is made out with invalid username and/or invalid password
- the login form is made out with valid username and valid password, so that user's account becomes temporally suspended
- a user tries to log in a suspended account
- a user answers correct each of the two questions, so that her account becomes unsuspended
- a user fails to answer one of the two questions
- a user answers one of the two questions in a time interval longer than 60 s.

This idea could be implemented in real webmail services, such as Hotmail [6] and Google mail [7]. In a real implementation, the list of valid e-mail addresses can be extracted from sent-mail list. The set of invalid e-mail addresses can be constructed using an algorithm that produces plausible e-mail addresses.

4.2 A PACT Enabled E-Commerce Service

For our second case study, we chose to implement our new security measure on a bidding site.

In this case study we wanted to prevent the misuse of a bidding account, even if a prior phishing attack was successful, and the malicious user had obtained a valid username/password pair for entrance in the authentic site.



Hello Alice

In order to proceed, please choose an e-mail address, in which you have sent an e-mail in the past:

parientecristina@yahoo.com
sotkat49@yahoo.com
criseze@terra.com.br
adimoe@yahoo.com
stanka70119@yahoo.com
zioyt1@yahoo.com
abn_jas@yahoo.com
gznidarec@yahoo.com.au
bert567@btopenworld.com
risitekton13@yahoo.com

Submit

Hello Alice

In order to proceed, please choose the item, that you have bought in the past:

Sony Vaio Laptop
Alienware Laptop
telephoto lens
Martial Arts Shoes
New Dell TFT Monitor
Accordeon
FileMaker Pro Software
Norton Internet Security
Antique Compass
Monitor Wall Mount
Antique Book

Submit

Fig. 1. Screenshots from the two prototype PACT enabled services deployed.

The PACT used in this case study is: "Select which item(s) you have bought in the past". The list of items, from which Alice has to pick the correct one, is dynamic, meaning that both the items and their position on the list, change every time Alice's logs in. The knowledge of an item bought at the past, is one that, theoretically, only Alice possesses. For the users that have not yet bought an item, we supplied them with a random "dummy purchase" at the time of their registration, so that they too, can be protected by our extra security measure. If the user trying to login, chooses the wrong item, or makes more than 60 s to decide, the account is suspended. Our core for the experiment is pretty much the same PHP pages and SQL tables used in our first case study of the e-mail service. In the right frame of Fig. 1, we present a screenshot of our bidding site.

4.3 Results

In this section, we present the experimental evaluation of our prototype implementation of Past Activity Tests (PACTs). We created two fake accounts, in Flickr [8] and in Google mail [7]. Also, two accounts were created, for our two case studies (the e-mail service and the bidding service). All of the above accounts correspond to a specific user, thus the username and password are the same. The details of these accounts were stored in a text file, which was shared in the Gnutella P2P network [9]. The purpose of this action was to invite attention of prospective attackers, in order to test our services' implementation. Respecting to the e-mail service, two specific users tried to log in, but in the first question, an invalid e-mail address was selected. In the bidding service, the same users tried to log in plus another one. The first user successfully logged in, with his first effort.

However, in his second try, a wrong item-product was selected. Then, the attacker tried again to log in, but his account was suspended, so the bidding service could not efficiently process his requirement. Concerning the second attacker, he picked an invalid product, thus his account remained suspended. Lastly, the third attacker tried to log in four times. His first try was successful but the rest were not. The three attackers used the pair of username and password, stored in the shared text file, as mentioned above.

5 Related Work

There are many approaches previously proposed for fighting phishing attacks.

The first approach is a browser plug-in, PwdHash [10], that transparently transforms user passwords to domain specific ones. This is done by substituting user's password with a hash of the password and the domain name of the web page that the password is going to be submitted. A main problem of this approach is the implementation in web sites that have multiple domain names.

A second approach is to use dynamically generated virtual skins [11]. Alice may customize her index page in Bob's site and store her preferences to Bob's database. Thus, an attacker can not clone Alice's index page in Bob's site, since he has not access to Alice's preferences, which are stored in Bob's database. However, this approach requires the developing of new habits from users, such as customization of the Internet services they use everyday.

AntiPhish [12] tries to protect users from giving away their sensitive data to untrusted web sites. Users have to specify what they consider sensitive data and which web sites are trusted. However, situations where an attacker tricks Alice to submit her credentials via another media, for example via e-mail, and not in a cloned web site can not be defeated by AntiPhish.

Another interesting approach, illustrated in [13], is based on visual similarity. A legitimate owner of a web page can use this technique to search the Web for suspicious web pages, which are visually similar to the true web page. The visual similarity between two web pages is based on three metrics: block level similarity, layout similarity and overall similarity. A web page is considered as a phishing suspect if any of these similarities to the true web page is higher than a threshold.

An approach, similar to PACTs, is the use of graphical passwords instead of text-based passwords. In an authentication procedure a user is presented with a set of images and he passes the authentication by recognizing and identifying the images he had selected during the registration stage. "Passface" is a technique, based on graphical passwords, developed by Real User Corporation [14]. Through this technique a user will be asked to choose four images of human faces from a database as his future password. In the authentication stage, the user sees a grid of nine faces, consisting of one face previously chosen and eight decoy faces. The user recognizes and clicks on the known face. This procedure is repeated for several rounds, thus the user is fully authenticated if he correctly identifies the four faces. Passfaces replaces or works in conjunction with text passwords. However, passface-based login process takes longer than text passwords. Also, another drawback is that passwords created using this technique have obvious patterns among them. This makes the passface-based password somewhat predictable.

Apart from the above, there is a series of academic papers [15,16], which try to detect phishing Web sites based on visual similarities or content based similarities [17]. Past Activity Tests (PACTs) is a very interesting but completely different approach to fight phishing.

6 Conclusions

In this paper we presented Past Activity Tests (PACTs) as a countermeasure against phishing attacks. PACTs rely on the idea that a user has accessed her account in the past, but an adversary is accessing it for the first time. Thus, a user can answer easily a question in regards to her past activity, but the adversary can not.

We presented two real deployments of PACT enabled on-line services. We created accounts for our on-line services and injected their credentials in P2P file sharing networks so as to lure possible adversaries to access our on-line services. The results showed that PACT could resist in an attempt from a third party trying to access the service with stolen credentials.

PACTs are not only an anti-phishing solution, but also a more secure way of general web-based authentication.

Part of our future work is to explore user-friendly PACTs for various on-line services which are not covered in this paper.

References

[1] APWG. Anti-phishing working group. http://www.antiphishing.org
[2] PHP. Hypertext preprocessor. http://www.php.net
[3] PostgreSQL. The world's most advanced open source database. http://www.postgresql.org

[4] Yahoo people search. http://people.yahoo.com
[5] Google. Search engine. http://www.google.com
[6] Hotmail. Free e-mail service with security by Microsoft. http://www.hotmail.com
[7] Gmail. A new kind of webmail. http://mail.google.com
[8] Flickr. Photo sharing service. http://www.flickr.com
[9] Gnutella. http://www.gnutella.com
[10] B. Ross, C. Jackson, N. Miyake, D. Boneh, J. C. Mitchell (2005). Stronger Password Authentication Using Browser Extensions. In SSYM'05: Proceedings of the 14th Conference on USENIX Security Symposium, pp. 2–2, Berkeley, CA, USA. Usenix Association
[11] R. Dhamija and J. D. Tygar (2005). The Battle Against Phishing: Dynamic Security Skins. In SOUPS '05: Proceedings of the 2005 Symposium on Usable Privacy and Security, pp. 77–88, New York, USA, 2005. ACM Press
[12] E. Kirda and C. Kruegel (2005). Protecting Users Against Phishing Attacks with AntiPhish. In COMPSAC '05: Proceedings of the 29th Annual International Computer Software and Applications Conference (COMPSAC'05) Vol. 1, pp. 517–524, Washington, DC, USA. IEEE Computer Society.
[13] L. Wenyin, G. Huang, L. Xiaoyue, Z. Min and X. Deng (2005). Detection of Phishing Webpages Based on Visual Similarity. In WWW '05: Special Interest Tracks and Posters of the 14th International Conference on World Wide Web, pp. 1060–1061, New York, USA. ACM Press
[14] Passfaces. Patented graphical passwords for enterprise. http://www.passfaces.com
[15] A.Y. Fu (2006). Detecting Phishing Web Pages with Visual Similarity Assessment Based on Earth Mover's Distance (EMD). IEEE Trans. Dependable Secur. Comput., 3(4):301–311. Senior Member-Liu Wenyin and Senior Member-Xiaotie Deng.
[16] W. Liu, X. Deng, G. Huang and A.Y. Fu (2006). An Antiphishing Strategy Based on Visual Similarity Assessment. IEEE Educational Activities Department, 10(2):58–65
[17] Y. Zhang, J. Hong and L. Cranor (2007). CANTINA: A Content-Based Approach to Detecting Phishing Web Sites. In Proceedings of the 16th International World Wide Web Conference (WWW2007)

8
QuiGon: The First Tool Against Clone Attack on Internet Relay Chat

Thibaut Henin and Corinne Huguennet

arsouyes.org, France
{Tbowan, aryliin}@arsouyes.org

Abstract. Clone attacks are a way to perform DDOS attack on Internet Relay Chat networks. There is no tool which prevents an IRC network against such attacks despite some handcrafted methods which are not so efficient. Here, we present qui-gon, the first tool which defeat clones attacks. It uses a temporal oracle which detect attacks by measuring the connection rate; and another oracle which distinguish clones using probabilistic automata. These oracles learn the characteristic of the network in order to fit better to the network.

1 Introduction

Deny Of Services Attack (DOS) is a kind of easy yet powerful attack. The aim of such attack is to make a system, a service or a website not working as it should. This can be done by sending some handcrafted packets or too many request to the server. This method is quite popular and simple [1–3].

When they are distributed, it's more difficult to detect and prevent them. Such attack is performed by compromising a lot of hosts and then order them to attack any service. Because of the many-to-one aspect of this kind of attack, common techniques can not be applied [1, 2].

These Distributed Deny Of Services (DDOS) also occurs on Internet Relay Chat and are called clone attacks [3]. The attack is performed by connecting lots of clones to a targeted server. The effect range from a server crashing down to networks split.

Here, we present Qui-Gon, the first tool against clone attacks on Internet Relay Chat servers. Despite two handcrafted methods, there is no released

V. Siris et al. (eds.), *Proceedings of the 3rd European Conference on Computer Network Defense*, Lecture Notes in Electrical Engineering,
DOI 10.1007/978-0-387-85555-4_8, © Springer Science+Business Media, LLC 2009

tool against such attack. Our method can detect the attack and recognize clones.

In the first part, we will present Internet Relay Chat and the clone attack. Next, we will explain some usual protections. We will then introduce Qui-gon and the methods used to detect the attack and clones. Finally, we give some test results and a conclusion.

2 Clone Attack on Internet Relay Chat

2.1 Internet Relay Chat

Internet Relay Chat (IRC) is a form of real time communication [4]. It's mainly used for group and broadcast communications. Once connected to a server, users join so-called channel where they can communicate with the users present in this channel. IRC is also used for private communications, file sharing and games.

Users are identified by their nicknames. Once connected to a server, a user chooses his nickname and sends it to the server. After that, he can join channel, speaks with others, and so on. Since many users can use the same public IP address, it can not be used to identify any user.

An IRC network is made by connecting servers to form a tree. Each server is responsible of some users and channels and routes any information to each concerned server. This network architecture was chosen to avoid duplication but, obviously, a server crash cause a network split. This lead to two disconnected IRC networks with duplicated channels.

In addition of simple users, there are so called IRC-operators. These users have privileges on the entire IRC networks and the servers. Their job is to administrate the network and keep it working. For example, they can disconnect users, ban users and IP addresses.

The first user who joins a channel gains the ownership of this channel. This first user is called the founder of the channel. The founder gains the most privileges on the channel and becomes a channel operator and can give same privileges to others users on the channel.

If a channel becomes empty, the first user to re-join the channel gains the ownership.

2.2 Bots and Botnets

Bots (abbreviation of "robots") are programs connected to any network [5, 6]. They perform simple and repetitive tasks such as searching the web and indexing web-sites. They can also replace players in online games.

On IRC, these bots are very commons. There are service bots, whose purpose is to maintain some order [7] such as NickServ which register and protect nicknames. Some bots provides more general features such as weather-reports, quiz-games or conversation finding [8]. Finally, there are some bots trying to pass the Turing test by speaking with users like ALICE [9].

Botnets are mostly networks formed by IRC bots which are waiting for commands [10]. These bots are installed in compromised hosts and connect themselves to an IRC network and join a specified channel where their Master can order them some actions. These actions vary from compromise others hosts, update their code and make a DDOS attack against another host [5, 6].

2.3 Clone Attacks

A clone attack is when an attacker mastering a botnet orders his bots to DDOS a server of the IRC network. The bots make multiple connections to the server (these connections are called “clones”) [3, 5, 6]. Because for each connection, the server keeps lots of informations, there is no need for too many clones to crash down a server.

Clone attacks are used to shutdown an IRC network, or to take-over an IRC channel. This kind of attack is an easy way to crash down a server. So, it’s an easy way to disturb users and IRC-operators. Once the network split occurs, the attacker may join some channels and gain ownership.

These clone attacks can’t be defeat only by IRC-operators. A clone attack is too brief and IRC-operators don’t have time to respond it. These attacks are also too intense. So, IRC-operators notice the attack only after its effect.

There are also clone attacks against channel instead of servers. In this case, the bots are already connected to an IRC server. When the attacker asks them, they simply join a specified channel and some time floods it (by sending lots of messages). This channel is then unusable.

This kind of attack can easily be defeat and is then out of the scope of this article. To avoid bots to send message on a channel, one can simply set the mode “m” (moderated) to the channel. This mode allow sending message only to users with the mode “v” (voice). This way, privileged users can set this mode to legitimate users they know manually, or ask chanserv to do this automatically.

Thus, in the next sections, we will only focus on clone attacks against servers.

2.4 How to Prevent Such Attacks?

Obviously, IRC-operators need an automated tool to detect, prevent and defeat clones attacks. Such tool should act just after the connection of any client and before his first join of any channel. This way, when a Clone Attack occurs, the tool will refuse any clone connection as early as possible and keep the network working. This tool need to implement in some way the three following features :

Detect the attack. Clone attacks occur a few times but are too brief and can not be detected by IRC-operators. So, tools who want to defeat such attacks need to detect them before their effects occur.

Distinguish clones by their nicknames. Once the attack is detected, we need to refuse clones but accept legitimate users. One could simply refuse all connections during the attack; however, this would prevent anybody from joining the network creating another kind of DOS [11]. As mentioned before, the only way to recognize users is their nickname. So, our tool needs to distinguish legitimate users by reading their nickname in order to accept them and refuse the clones.

Learn. Each IRC-network has its own characteristics. A connection which could be considered as an attack in a small network should be considered as normal traffic in a bigger one that may have a lot of connection per hour. Because of different cultural habits between networks, nicknames differ between networks [12]. A nickname which sounds like clones in one network could be one of a legitimate user in an other.

3 Usual Protections

We have found no publication or tools concerning the defence against clone attacks despite one feature of IRC networks and two handcrafted methods. These methods have been seen in some IRC networks but have not been released.

3.1 Passwords to Enter the Network

Access to IRC networks can be protected by a password. This way, when a user wants to connect to a server, he must give the good password. Such networks are called private networks. While this technique is effective to protect the network against unguests users, it obviously does not prevent it against clones mastered by a user of this network knowing the pass.

3.2 Blacklist

This method distinguish clones from their IP addresses. The main idea is to keep a black-list of IP Address. These IP addresses are manually added to the list after an attack and send to other IRC-operator so they could use the list for other IRC networks.

Obviously, this is not an effective method to defeat a Clone Attack. As already mentioned, many users can use a single IP Address. If this address has been added to the black-list, legitimate user will be refused by the system. Furthermore, new bots, whose address is not already in the list, will be accepted.

3.3 Use Simple Regexp

Since clone nicknames are automatically generated, the second method use simple regexp[1] to distinguish clones. The regexp is made by IRC-operators who find some characteristics in the nicknames of the clones after an attack. For example, any nickname ending with two digits can be considered as a clone nickname.

Again, this method is not effective. A simple regexp is too restrictive and lots of legitimate nicknames are considered as clones' nicknames. In addition, when the regexp is known, it is very easy to create nicknames that match the regexp.

4 Qui-Gon

Qui-Gon is our tool against clone attacks. It detects attacks, distinguishes clones and learns to fit better his network.

Its architecture is based on two oracles. The first one (the temporal oracle) detects attacks and when asked, answers whatever or not an attack occur. The second oracle (the distinguishing oracle) distinguishes clones and users. This way, we can improve each oracle individually and if another method to detect attacks or distinguish clones is found, it can be incorporate simply by replacing the oracle.

Both oracle are working together to improve the system. In fact, when the temporal oracle detect a false attack (to much users connecting themselves), the distinguishing oracle let them connect because it recognize them. Furthermore, the only negatives false the distinguishing oracle can make are while a detected attack and won't be embarrassing.

[1] A regexp: regular expression [19].

4.1 The Temporal Oracle

To detect an attack, the oracle measures the connection rate. As already stated, an attack consists of a lot of connections during a short time. Thus, during an attack, the connection rate increases significantly and then can be detected.

The oracle considers an attack occurs when the average rate exceed a learned value. We can simply measure the time between two connections. But if two users connect themselves at the same time, the oracle see an attack. Thus, we take the time taken by a fixed number of connections as main measure. The learn phase take place in peace period; the oracle observed the maximum value and take it as reference.

Since the connection rate is very fast, this method is effective. Using a large number of connections, we are able to smooth the impact of few simultaneous legitimate connections. Although, with a large number of connection, we will not refuse the first clones but all the next ones. Despite the few clones accepted, the others will not crash down the server and the attack will be defeat.

4.2 The Distinguishing Oracle

As stated before, users from an IRC network use nickname which follows some rules [12, 13]. For cultural or habits reasons, users do not choose their nicknames as random but throughout some mental process [12].

Furthermore, clone nicknames are automatically generated and follow some computational rules. Usually, they are choosing randomly over the set of all possible nicknames made by letters and digits [14].

The main idea is to use probabilistic automata to distinguish clones and users.

Probabilistic automata used to distinguish clones Distinguishing clones could be done with Markov chains. Markov chains are well suited for classification problems such as distinguishing set of first names [15].

Instead of using directly Markov chain, we conceptualize our sets of clone nicknames and legitimate users with Probabilistic automata [16, 17]. There is no formal definition of probabilistic automata and we will use the one from Henin [16]. A probabilistic automaton is an automaton with probabilities on transitions and final states. To get the probability of the nickname, one simply read it throughout the automata's transitions. The probability of the nickname is the product of the probabilities on these transitions and the probability of the ending state to be final. This way, languages conceptualized by Markov chains are included into those conceptualized by our automata [16].

The distinction is made by comparing the probability given by two automata. The first automaton (called the bad one) identifies clones' nicknames while the second (called the good one) identifies legitimate users' nicknames. If the probability given by the good automaton is bigger than the one given by the bad automaton, the nickname is considered as a legitimate user's nickname.

The good automaton learns during peace period of time. Because a clone attack is always fast, during that phase, we consider all users are legitimate ones. The automaton uses then all these nicknames as forming the learning set.

The bad automaton learn only on demand. Because of a lot of users during the clone attack are not all clone, we can't learn automatically. After an attack, the IRC-operators must sort nicknames from the attack. The set of nicknames considered as clones is then given to the bad automaton. These set of clones names can be sent to other IRC-operators of other networks to prevent them for such attack.

5 Tests and Validation of the Distinguishing Oracle

We have tested the distinguishing oracle with some sets of nicknames. We have used a set of 259 nicknames found in [12] (set A1), another set of 107 nicknames found on the IRC network geekirc[2] (set A2). These are real nicknames of human users. We have also used clones nicknames: a set of 70 nicknames generated by a real tool performing clones attacks [14] (set B1) and another set of 20 nicknames from a real attack we have saw on geekirc (set B2).

The set B2 from the real attack does not follow the same pattern as the set generated by the tool (set B1). The tool generates nickname by choosing 9 random letters. The nicknames from the real attack are also chosen randomly and are formed with 5 letters and 1–3 digits append to the end. Both pattern are use to perform real clone attacks and are not just artificial patterns.

Here, we show two scenarios of tests. The first one tests the oracle against existing attacks and tools. Next, we present a more clever attack.

[2] The authors were IRC operators of this IRC network.

5.1 Test Against Existing Attacks

In this test, the oracle has learned a subset of 200 nicknames of A1 as legitimate ones and a subset of 50 nicknames of B1. Theses sets represent a set of users the oracle have already seen on the networks and some nicknames of bots it has also seen.

Then, we have tested the oracle on all the sets of nicknames. In this test, we consider that some new human user want to connect to our networks while an attack. This attack consists of clones generated by some real tools (sets B1 and B2). Results are shown on Table 1.

Firstly, we notice that the oracle made no negative false. As we can see (Table 1), all the clones are well recognized. The oracle considered as clones all those that have been learned and all those generated by the same pattern. It shows that the learning process is effective. Furthermore, the 20 clones from geekirc were also well recognized showing us that the oracle can recognize unknown clones made by other tools.

Next, There's little positive false in the learning set. There were only 3 nicknames badly recognized. So, during an attack, the oracle will let connect almost all legitimate users it has already seen.

Finally, new legitimate users could sometime connect while an attack. As we can see, more than a half of the nicknames from the same network as the learning set (A1) are well recognized. One out of three nicknames from another network (A2) is well recognized. Thus, the oracle let connect some legitimate new users and refuse some others. It's not a bad point since the distinction is made only during an attack. The QoS[3] will be maintained. Finally, as we can infer [12, 13], users from the same background are better recognized than those from another background.

Table 1. Classification of nicknames into legitimate users' nicknames. After learning 200 legitimate nicknames and 50 generated by a real clone attack tool each set has been tested. For each set, we give the amount of nicknames considered as legitimate users and the amount of errors the oracle has done.

Population	Considered as legitimate
200 Legitimate learned nicknames (subset of A1)	98.5 %
59 Legitimate tested nicknames (subset of A1)	57.6 %
107 Legitimate tested nicknames (set A2)	36.4 %
50 Learned Clones from pru.c (subset of B1)	0.0 %
20 Tested Clones from pru.c (subset of B1)	0.0 %
20 Tested Clones from a real attack (set B2)	0.0 %

[3] Quality of Service.

Table 2. Behaviour against more clever attacks. After learning A1 as legitimate nicknames and 50 nicknames from A2 as clone ones each set has been tested. For each set, we give the amount of nicknames considered as legitimate users and the amount of errors the oracle has done.

Population	Considered as legitimate
259 Legitimate learned nicknames (A1)	91.1 %
50 Learned Clones from A2	4.0 %
57 Tested Clones from A2	42.0 %

5.2 Test Against a More Clever Attack

Obviously, one can also pick some nicknames from others networks, websites, dictionaries ... In this test, we consider the set A1 of nicknames from geekirc as the picked ones and use them as nicknames of clones. As the previous test show, the first time this attack occurs, only 36 percent of clones will succeed.

But once the first attack have finished, QuiGon learn the set of clones and things may change. Thus, in this test, the oracle has learnt the set A1 as legitimate users and a subset of 50 nicknames of A2 as clones. Then, we have tested the oracle on the set A1 and A2. Results are shown in Table 2

First, we can see that the oracle still make few positive false. Only around 10 percent of legitimate users have been considered as clones. But since this occurs only while an attack, these legitimate users will be able to connect themselves later, which is not so embarrassing.

Next, we can see that this time, the oracle make some negative false but learn efficiently. He have considered as legitimate 4 percent of clones it has already seen and 42 percent of clones it never saw. Once QuiGon has learnt a set of nicknames of clones, he will recognize them as clones later if they come back.

6 Conclusions

By using Probabilistic automata, the tool can learn the nicknames patterns and adapt itself to other networks. The probabilistic automata are well suited to classified nicknames [17] and produce no negative false. Despite some positive false, the oracle recognizes almost all legitimate users it's already seen.

Its use two distinct oracles, both could be replace and allow to simply improve the tool. We are looking to improvements. Obviously, the network activity is not constant and we're trying to conceptualize such characteristic into the temporal oracle. Furthermore, the distinguishing oracle know a fixed automata. Despite the probability on the transitions and stats, the structure can't evolve. We're trying to adapt state melting [16] and state splitting [18] to those automata in order to characterize better the nicknames sets.

Thus, QuiGon is the first and only tool dedicated to prevent clone attacks. Despite some handcrafted methods which are not so efficient there are no such tools. Our method can detect attack and distinguish clones. This way, during an attack, the clones are refused, defeating the attack.

References

[1] J. Mirkovic, P. Reiher: A Taxonomy of DDoS Attacks and Defense Mechanisms, ACM SIGCOMM Computer Communications Review, vol. 34, no. 2, 2004
[2] C. Douligeris, A. Mitrokotsa: DDoS Attacks and Defense Mechanisms: Classification and State-of-the-Art, Computer Networks, vol 44, no 5, 2003
[3] H. Alaganandam, P. Mittal, A. Singh, C. Fleizach: Cybercriminal Activity, 2005
[4] J. Oikarinen, D. Reed: Internet Relay Chat Protocol, IETF, RFC n1459, 1993
[5] B. McCarty: Botnets: Big and Bigger, IEEE Security and Privacy, vol. 1, no. 4, 2003
[6] D. Geer: Malicious bots threaten network security, IEEE Computer Society, vol. 38, no. 1, 2005
[7] D. R. Karrels: Internet Relay Chat Service Framework : GNUWorld, 2003
[8] N. W. Van Dyke, H. Lieberman, P. Maes: Butterfly: a conversationfinding agent for Internet Relay Chat, 4th International Conference on Intelligent User Interfaces, 1998
[9] Dr. R. S. Wallace: The Elements of AIML Style, A.L.I.C.E. Artificial Intelligence Foundation, Inc., 2003
[10] J. Canavan: The evolution of malicious IRC Bots, Virus Bulletin Conference, 2005
[11] D. Bernstein: Syn cookies - http://cr.yp.to/syncookies.html, Last accessed: July 2007
[12] H. Bechar-Israeli: From Bonehead to cLoNehEAd: Nicknames, Play and Identity on Internet Relay Chat, Journal of Computer-Mediated Communication, vol 1, no. 2, 1996
[13] E. M. Reid: Electropolis : Communication and Community on Internet Relay Chat, 1991
[14] http://packetstormsecurity.org/DoS/pru.c, Example of clone attack tool, Last accessed: July 2007
[15] P. Dupont: Noisy Sequence Classification with smoothed Markov Chains, 8me confrence francophone sur l'apprentissage automatique, 2006
[16] T. Henin: Reprsentation par jeu du chaos de squences d'ADN, ENS Cachan, University Rennes 1, 2007

[17] F. Thollard, A. Clark: Apprentissage d’automates probabilistes dterministes, Confrence d’Apprentissage, 2004
[18] J. Callut, P. Dupont: Inducing Hidden Markov Models to Model Long-Term Dependencies, Artificial Intelligence, European Conference on Machine Learning, 2005
[19] V. Laurikari: Efficient Submatch Addressing for Regular Expressions, Helsinki University of Technology, 2001

9
Defending Against Next Generation Through Network/Endpoint Collaboration and Interaction

Spiros Antonatos[1], Michael Locasto[2], Stelios Sidiroglou[2], Angelos D. Keromytis[2] and Evangelos Markatos[1]

[1] Foundation for Research and Technology Hellas, Heraklion, Greece
{antonat, markatos}@ics.forth.gr
[2] Department of Computer Science Columbia University, New York, USA
{locasto, stelios, angelos}@cs.columbia.edu

1 Introduction

Over the past few years we have seen the use of *Internet worms,* i.e., malicious self-replicating programs, as a mechanism to rapidly invade and compromise large numbers of remote computers [30]. Although the first worms released on the Internet were large-scale, easy-to-spot massive security incidents [6, 17, 18, 23], also known as *flash worms* [29], it is currently envisioned (and we see already see signs, in the wild) that future worms will be increasingly difficult to detect, and will be known as *stealth worms*. This may be partly because the motives of early worm developers are thought to have been centered around self-gratification brought by the achievement of compromising large numbers of remote computers, while the motives of recent worm and malware developers have progressed to more mundane (and sinister) financial and political gains. Therefore, although recent attackers still want to be able to control a large number of compromised computers, they prefer to compromise these computers as quietly as possible, over a longer period of time, so as to impede detection by current defense mechanisms. To achieve stealthy behavior, these attackers have started using, or at least have the capacity to use, a wide variety of mechanisms that will make their worms more difficult to detect. Such mechanisms might include:

- **Encryption:** Attackers may communicate with the potential victim using a secure (encrypted) connection, making it difficult for network-based Intrusion Detection Systems [22, 32] to spot their attempted attack.
- **Metamorphism:** The body of worms usually contains some initial code that will be executed when the worm invades the victim computer. Metamorphism obfuscates this code by adding various instructions to it, and/or by substituting blocks of instructions with equivalent blocks of other instructions [31]. In this way, two "copies" of the worm would appear to be completely different from each other, confusing worm detection systems that depend on all copies of a worm being practically identical [1, 14, 27].
- **Polymorphism:** Polymorphic approaches obfuscate the worm's body by encoding it and prepending a decoder. When propagating, the worm mutates its body so that two "copies" of the worm would look completely different from each other (modulo the body of the encoder) [10, 15, 31]. Much like metamorphic approaches, polymorphic systems confuse worm detection systems.
- **Hit Lists:** The first versions of recent worms selected their victims pseudo-randomly, *i.e.,* by generating a random IP address in the range *0.0.0.0* to *255.255.255.255*. It has been proposed however, that worms may be more effective if they first create a hit-list of all vulnerable computers and then attack only computers in that hit-list [3, 30]. This hit-list may even be filtered to exclude honeypots.[1] Armed with a hit-list, a worm is able to compromise a number of vulnerable computers, while generating the minimum amount of traffic possible, limiting the effectiveness of defense mechanisms that detect visible traffic anomalies.
- **Hybrid Worms**: Traditionally, worms have exploited vulnerabilities in applications and services open to Internet traffic. However, as more computers are located behind firewalls and NATs, they are theoretically protected from such types of attacks. Unfortunately, worm developers may exploit several different invasion paths including, infected email attachments, infected files shared through peer-to-peer (P2P) networks, and infected files accessed through locally shared disks [13].

[1]A honeypot is a computer waiting to be attacked. Once attacked, the honeypot records as much information as possible so that the administrators will be able to characterize the attack and possibly generate a signature for it.

- **Defense Mapping:** Many of the proposed (and deployed) techniques for detecting and countering new attacks use honeypots as early-warning systems [5, 8, 9, 20, 28, 33]. However, recent work has shown that attackers can exploit certain features and aspects of honeypot behavior to identify and avoid such detectors [7, 21, 24]. Combined with hit-lists, this can render worms (especially slow-spreading ones) and other automated attacks virtually undetectable.
- **Client-side Attacks:** In the past few years (2005–2006) we have seen an increase in the use of zero-day attacks aimed at client software (especially browsers, but also various types of document viewers such as Microsoft Word, Excel and PowerPoint, and Adobe Acrobat). Other than stand-alone, host-based intrusion detection/prevention mechanisms (such as virus scanners), very little has been done in hardening vulnerable client systems.

1.1 Impact of Failing to Solve the Problem

Compromised computers can be used to cause harm to third parties or even to cause harm to their traditional owners.

- **Attacks to third parties:** Recent worm writers organize compromised computers into botnets, i.e., armies of hosts that are primarily used for malicious acts, including launching of Denial of Service (DoS) attacks, blackmailing, sending of SPAM mail, click fraud, theft of intellectual property, and even identity theft. One would envision that botnets in the future could be used for political warfare purposes as well.
- **Attacks to the owners of compromised computers:** A compromised computer can be used to steal private data and facilitate identity theft. Unfortunately, once ordinary users start to realize the dangers of a compromised computer, they will probably get increasingly less inclined to trust their computers for financial transactions or private communications. This will probably impede the adoption of an information society and may eventually reduce its overall spread and impact.

2 Research Directions

Over the last five years significant research has been conducted in the area of detection and containment of cyber-attacks. Indeed, we believe that we have currently reached the point where it is possible to readily detect one particular class of worms: rapidly spreading and massively parallel flash worms. However, it is unclear we have the technical knowledge or the deployed mechanisms in order to detect and contain *stealth attacks*. Using a combination of the techniques described earlier, such attacks can become invisible (or at least very difficult to detect) to network-based defenses.

Our view is that such attacks can only be detected via large-scale collaboration among end-hosts: by exchanging and correlating relevant information, it is possible to identify stealthy attacks, and to take appropriate measures to defend against them, or at least quarantine those nodes that appear to have been compromised. Specifically, we believe that it is increasingly important to include home and small business computers in the attack-detection process. These computers are increasingly becoming the primary targets of most attackers. Therefore, including them in the worm- (or, more generally, attack-) detection process will increase the chances of attack detection. Exemplifying a large range of access patterns and a large range of applications, these computers typically tend to have more representative configurations than the traditional honeypots currently being used in worm detection. Furthermore, ordinary computers being used by their regular owners are more difficult to be categorized as honeypots and avoided by future attacks. The inclusion, however, of home computers in the detection process, should *(1)* guarantee the safety of the end computer and *(2)* the minimum possible intrusion in the ordinary use of the computer. Towards this direction, we propose two systems: Honey@home and Application Communities. We give a high-level description of both systems in the next two sections, both as concrete examples of collaborative defense mechanisms and to motivate further work in this direction.

On the other hand, we are not completely discounting network-based defenses: rather, we believe that such defenses must be integrated with end-host defenses. In the past, network and end-host security were viewed as two distinct areas that were meant to complement each other but kept separate. While this allowed for a clean separation between the respective security mechanisms, it also meant that the potential of both was stunted. Furthermore, by keeping them isolated, it was (and is) impossible to exploit scale for defensive purposes. Exploiting scale is something that attackers have learned to do well, as evidenced by such phenomena as distributed denial of service attacks, self-propagating worms, and botnets.

The industry is beginning to follow such an approach, albeit in a fragmented, *ad hoc* fashion. For example, several enterprises exchange alert and IDS logs through sites such as DShield.org; anti-virus vendors with extensive presence on the desktop are correlating information about application behavior from thousands of hosts; network security and monitoring companies perform similar correlation using network traces and distributed black-holes (honeypots). To the extent that such approaches are being explored, they seem largely confined to the realm of information gathering. This also largely seems to be the situation with the US Department of Defense and the various intelligence agencies. For example, DARPA is currently funding the Application Communities effort, which seeks to leverage large software monocultures to distribute the task of attack monitoring – again, an approach confined to the end-host. Previous work (notably in the DARPA OASIS program) looked into the space of reactive security, but only considered small-scale environments. Arguably, we need to extend the reach of our collaboration-based mechanisms to counter such pervasive threats as DDoS and botnets.

Thus, we argue that it is important to transition into an network architecture design where networks and end-hosts, in various combinations, can elect to collaborate and coordinate their actions and reactions to better protect themselves (and, by implication, the network at large). There are several research issues arising in such an environment, including:

- What problems are best addressed through a collaborative approach;
- New mechanisms at all levels of the network architecture (routers, protocols, end-hosts, processes, hardware) that are "collaboration friendly";
- Metrics that quantify the security of collaborative approaches over non-collaborative approaches ;
- Who to trust, and to what extend;
- How to prevent attacks that exploit such mechanisms, including insider threats;
- Command-and-control *vs.* loose-coupling mechanism composition.

Furthermore, in an era of distributed software services (what is fashionably called "Web 2.0"), no single application, node, or network has enough information to detect and counter high-level semantic attacks, or even some of the more conventional web-based malware (e.g., cross-site scripting attacks). Large-scale distributed systems require large-scale distributed defenses. This is particularly true within specific application domains (such as health care and industrial SCADA control), where large-scale collaborative (but independent) defenses will allow better control to critical information and resources.

3 Honey@home

Traditional honeypot architectures are based on monitoring unused IP addresses located at specific institutes and organizations [8]. This unused IP address space, also called “dark space”, is easy to identify and thus be blacklisted by attackers [7]. Furthermore, all honeypot technologies rely on the size of the dark space in order to be effective; the more dark space is used, the faster and more accurate the results obtained. To overcome these two problems, Honey@home [4] empowers ordinary users and organizations, institutes and enterprises, who are not familiar with honeypot technologies, to contribute their dark space to a network of affined honeypots. Many public bodies, universities and even home users do not use all the address space they possess. They also do not have the expertise to setup and maintain a honeypot to monitor that unused space. Honey@home fills that gap by installing a virtual honeypot to the machine(s) of unfamiliar users. Several other “@home” approaches, like Seti@home and Folding@home, have shown that users can contribute significantly towards a common goal.

Honey@home is designed to be used by people unfamiliar with honeypot technologies. From the user perspective, no configuration is needed. Honey@home is a cross-platform tool that requires minimal resources and can run unsupervised at the background, just like modern messengers. Its basic functionality is to claim an unused IP address through the DHCP server of the local network it is installed on and forward all the traffic going to that address to a centralized farm of honeypots. The centralized farm runs multiple services/applications, and processes all the traffic received from Honey@home clients. Central honeypots will provide answers to the received traffic and send them back to the Honey@home clients. From their side, Honey@home clients will send the responses from honeypots back to the originators of the attack. The attacker is under the impression that she communicates with the address claimed by Honey@home client, but in reality she communicates with a central honeypot that gathers, analyzes, and responds to her attacks and probes. More advanced users can manually declare their dark space and contribute more than one unused IP address. The centralized farm is implemented by a number of Argos [19] honeypots that are able to catch previously unknown attack vectors.

Honey@home enables the creation of an infrastructure where the monitored dark space is distributed over the network and can become arbitrarily large, depending on the number of Honey@home clients. Although the idea of forwarding traffic destined for an unused IP address to a centralized farm of honeypots may sound simple, there are several challenges behind the Honey@home approach. First, participating clients should be undetectable. If an attacker can easily determine whether an address is monitored by Honey@home, clients can be blacklisted and not contribute to the overall infrastructure. (Note, however, that this could be turned into a defensive advantage by acting as a deterrent.) Second, central honeypots must be hidden so that they cannot be remotely exploited or otherwise attacked. Finally, the installation of mock clients that will overload the central honeypots with nonsense traffic must be prevented. Honey@home tries to deal with these challenges by employing various techniques, like anonymization networks [11] to hide honeypots and a registration process to prevent massive automatic installation of fake clients.

4 Application Communities

An Application Community (AC) is a collection of congruent instances of the same application running autonomously on end-hosts distributed across a wide-area network, whose members cooperate in identifying previously unknown flaws/attacks [16]. By exchanging information, the AC members may be able to prevent the failure from manifesting in the future. Although individual members may be susceptible to new failures, the AC should eventually converge into a state of immunity against a particular fault, adding a dimension of learning and adaptation to the system. An AC may be considered a “virtual honeypot” composed of many machines/applications that are in actual use (i.e., they are not passive, non-guided entities as traditional honeypots are); AC members contribute a share of their resources (such as CPU cycles) towards the processing done by this virtual honeypot. By using real applications and systems as detectors, Application Communities can identify targeted attacks, attacks that exploit specific state, or attacks that require user action (e.g., for client applications such as web browsers). The size of the AC, in terms of number of participating nodes, impacts coverage (in detecting faults) and fairness (in distributing the monitoring task) [2]. An AC is composed of three main mechanisms, for monitoring, communication, and defense, respectively.

The purpose of the **monitoring mechanism** is the detection of previously unknown (“zero day”) software failures. There exists a plethora of work in this area, namely, using the compiler to insert run-time safety checks,“sandboxing”, anomaly detection, and content-based filtering [2]. While shortcomings may be attributed to each of the approaches, when they are considered within the scope of an AC a different set of considerations need to be examined. Specifically, the significance of the security *vs.* performance tradeoff is de-escalated with respect to the ability to efficiently employ the mechanism in a distributed fashion. The advantage of utilizing an AC is that the use of a fairly invasive mechanism (in terms of performance) may be acceptable, since the associated cost can be distributed to the participating members. By employing a more invasive instrumentation technique, the likelihood of detecting subversion and identifying the source of the vulnerability is increased. The monitoring mechanism in our prototype is an instruction-level emulator that can be selectively invoked for arbitrary segments of code, allowing us to mix emulated and non-emulated execution inside the same execution context [26], although other mechanisms can be used instead (or in addition) [25].

Once a failure is detected by a member's monitoring component, the relevant information is distributed across the AC. Specifically, the purpose of the **communication component** is the dissemination of information pertaining to the discovery of new failures and the distribution of the monitoring work load within the AC. The choice of the communication model to be employed by an AC is subject to the characteristics of the collaborating community, such as size and flexibility. The immediate tradeoff associated with the communication model is the overhead in messages versus the latency of the information in the AC. In the simplest case, a centralized approach is arguably the most efficient communication mechanism. However, there are a number of scalability and trust issues associated with this approach. If there is a fixed number of collaborating nodes, a secure structured overlay network can be employed, with exemption from the problems associated with voluminous joins and leaves. If nodes enter and leave the AC at will, a decentralized approach may be more appropriate. Efficient dissemination of messages is outside the scope of this paper, but has been the topic of much research in the networking community.

The **immunizing component** of our architecture is responsible for protecting the AC against future instances of a specific failure. Ideally, upon receiving notification of a failure observed by another AC member, individual members independently confirm the validity of the reported weakness and create their own fix in a decentralized manner. At that point, each member in the AC decides autonomously what fix to apply in order to inoculate itself. As independent verification of an attack report may be impossible in some situations, a member's action may depend on predefined trust metrics (e.g., trusted verifications servers). Depending on the level of trust among users, alternative mechanisms may be employed for the adoption of universal fixes and verification of attack reports. In the case of systems where there is minimal trust among members a voting system can be employed at the cost of an increased communication overhead. Finally, given that a fix could be universally adopted by the AC, special care must be placed in minimizing the performance implications of the immunization.

The inoculating approach that can be employed by the AC is contingent on the nature of the detection mechanism and the subsequent information provided on the specific failure. The type of protection can range from statistical blocking, behavioral or structural transformation. For example, IP address and content filtering, code randomization [12], adaptive defenses [25], and emulation [26] may be used for the protection of the AC members.

5 Conclusions

We have argued that the Internet-borne cyber-attacks of the future require collaborative solutions that encompass (and perhaps focus) on end-hosts, rather than depend on network-based defenses. We have briefly described two such research thrusts, Honey@home [4] and Application Communities [AC06]. Although there are many research challenges (and opportunities) ahead, we believe that large-scale collaborative defenses hold the key to a future secure Internet.

Acknowledgments

This material is partially based on research sponsored by the Air Force Research Laboratory under agreement number FA8750-06-2-0221 and by NSF Grant 06-27473, with additional support from Google and New York State. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. There opinions expressed herein do not reflect those of the NSF or the U.S. Government. This work was also supported in part by the project Cyber-Scope, funded by the Greek General Secretariat for Research and Technology under the contract number PENED 3ED440, and by the FP6 project NoAH, funded by the European Union under the contract number 011923.

References

[1] P. Akritidis, K. G. Anagnostakis, and E. P. Markatos. Efficient content based worm detection. In Proceedings of the 40th IEEE International Conference on Communications (ICC), 2005.

[2] K. G. Anagnostakis, S. Sidiroglou, P. Akritidis, K. Xinidis, E. Markatos, and A. D. Keromytis. Detecting targeted attacks using shadow honeypots. In Proceedings of the 14th USENIX Security Symposium, pp. 129–144, August 2005.

[3] S. Antonatos, P. Akritidis, E. P. Markatos, and K. G. Anagnostakis. Defending against hit list worms using network address space randomization. In Proceedings of the ACM Workshop on Rapid Malcode (WORM), pp. 30–40, November 2005.

[4] S. Antonatos, K. G. Anagnostakis and E. P. Markatos. Honey@home: A new approach to large-scale threat monitoring. To appear in the Proceedings of the 5th ACM Workshop on Recurring Malcode (WORM), November 2007.

[5] M. Bailey, E. Cooke, F. Jahanian, J. Nazario, and D. Watson. The internet motion sensor: A distributed blackhole monitoring system. In Proceedings of the 12th ISOC Symposium on Network and Distributed Systems Security (SNDSS), pp. 167–179, February 2005a.

[6] M. Bailey, E. Cooke, F. Jahanian, D. Watson, and J. Nazario. The blaster worm: Then and now. In IEEE Security & Privacy Magazine, 3(4):26–31, 2005b.

[7] J. Bethencourt, J. Franklin, and M. Vernon. Mapping internet sensors with probe response attacks. In Proceedings of the 14th USENIX Security Symposium, pp. 193–208, August 2005.

[8] E. Cooke, M. Bailey, Z. M. Mao, and D. McPherson. Toward understanding distributed blackhole placement. In Proceedings of the ACM Workshop on Rapid Malcode (WORM), pp. 54–64, October 2004.

[9] D. Dagon, X. Qin, G. Gu, W. Lee, J. Grizzard, J. Levine, and H. Owen. HoneyStat: Local worm detection using honepots. In Proceedings of the 7th International Symposium on Recent Advances in Intrusion Detection (RAID), pp. 39–58, October 2004.

[10] T. Detristan, T. Ulenspiegel, Y. Malcom, and M. Underduk. Polymorphic shellcode engine using spectrum analysis. In Phrack, 11(61), August 2003.

[11] R. Dingledine, N. Matthewson, and P. Syverson. Tor: The second-generation onion router. In Proceedings of the 13th USENIX Security Symposium, August 2004.

[12] G. S. Kc, A. D. Keromytis, and V. Prevelakis. Countering code-injection attacks with instruction-set randomization. In Proceedings of the 10th ACM Conference on Computer and Communications Security (CCS), pp. 272–280, October 2003.

[13] D. M. Kienzle and M. C. Elder. Recent worms: A survey and trends. In Proceedings of the ACM Workshop on Rapid Malcode (WORM), pp. 1–10, 2003.

[14] H. Kim and B. Karp. Autograph: Toward automated, distributed worm signature detection. In Proceedings of the 13th USENIX Security Symposium, pp. 271–286, August 2004.

[15] K2. ADMmutate. http://www.ktwo.cal/ADMmutate-0.8.4.tar. gz.

[16] M. E. Locasto, S. Sidiroglou, and A. D. Keromytis. Software self-healing using collaborative application communities. In Proceedings of the 13th ISOC Symposium on Network and Distributed Systems Security (SNDSS), pp. 95–106, February 2006.

[17] D. Moore, V. Paxson, S. Savage, C. Shannon, S. Staniford, and N. Weaver. Inside the slammer worm. In IEEE Security & Privacy Magazine, 1(4): 33–39, 2003.
[18] D. Moore, C. Shannon, and J. Brown. Code-Red: A case study on the spread and victims of an Internet worm. In Proceedings of the 2nd ACM SIGCOMM Workshop on Internet Measurement (IMW), pp. 273–284, 2002.
[19] G. Portokalidis, A. Slowinska, and H. Bos. Argos: An emulator for fingerprinting zero-day attacks. In Proceedings of ACM SIGOPS Eurosys, April 2006.
[20] M. A. Rajab, F. Monrose, and A. Terzis. On the effectiveness of distributed worm monitoring. In Proceedings of the 14th USENIX Security Symposium, pp. 225–237, August 2005.
[21] M. A. Rajab, F. Monrose, and A. Terzis. Fast and evasive attacks: Highlighting the challenges ahead. In Proceedings of the 9th International Symposium on Recent Advances in Intrusion Detection (RAID), pp. 206–225, September 2006.
[22] M. Roesch. Snort: Lightweight intrusion detection for networks. In Proceedings of USENIX LISA, pp. 229–238, 1999.
[23] C. Shannon and D. Moore. The spread of the Witty worm. In IEEE Security & Privacy Magazine, 2(4):46–50, 2004.
[24] Y. Shinoda, K. Ikai, and M. Itoh. Vulnerabilities of passive internet threat Monitors. In Proceedings of the 14th USENIX Security Symposium, pp. 209–224, August 2005.
[25] S. Sidiroglou, G. Giovanidis, and A. D. Keromytis. A Dynamic mechanism for recovering from buffer overflow attacks. In Proceedings of the 8th Information Security Conference (ISC), pp. 1–15, September 2005.
[26] S. Sidiroglou, M. E. Locasto, S. W. Boyd, and A. D. Keromytis. Building Ra Reactive Immune System for Software Service. In Proceedings of the USENIX Annual Technical Conference, pp. 149–161, April 2005.
[27] S. Singh, C. Estan, G. Varghese, and S. Savage. Automated Worm Fingerprinting. In Proceedings of OSDI, pp. 45–60, 2004.
[28] L. Spitzner. Honeypots: Tracking Hackers. Addison-Wesley, Boston, MA, 2003.
[29] S. Staniford, D. Moore, V. Paxson, and N. Weaver. The top speed of flash worms. In Proceedings of the ACM Workshop on Rapid Malcode (WORM), pp. 33–42, November 2004.
[30] S. Staniford, V. Paxson, and N. Weaver. How to own the internet in your spare time. In Proceedings of the 11th USENIX Security Symposium, August 2002.
[31] P. Szor and P. Ferrie. Hunting for metamorphic. In Proceedings of the Virus Bulletin Conference, pp. 123–144, September 200l.
[32] K. Xinidis, I. Charitakis, S. Antonatos, K. G. Anagnostakis, and E. P. Markatos. An active splitter architecture for intrusion detection and prevention. In IEEE Transactions on Dependable Secure Computing, 3(1):31–44, 2006.
[33] V. Yegneswaran, P. Barford, and D. Plonka. On the design and use of internet sinks for network abuse monitoring. In Proceedings of the 7th International Symposium on Recent Advances in Intrusion Detection (RAID), pp. 146–165, October 2004.

10
ISi-LANA – A Secure Basic Architecture for Networks Connected to the Internet

Thomas Haeberlen

Federal Office for Information Security (BSI), Bonn, Germany
thomas.haeberlen@bsi.bund.de

Abstract. This P-A-Per gives a brief summary of a study compiled for the German Federal Office for Information Security (BSI). The study will be published as part of a new series of documents on Internet security later this year. A basic architecture to support secure operation of a network connected to the internet is proposed. By implementing this basic architecture, the risks associated with connecting a network to the internet, can be greatly reduced. The basic architecture and recommendations cover the robust design of the network, the selection and configuration of network equipment, as well as aspects of network operations. The view is mostly on security aspects of the lower layers of the TCP/IP reference model; application specific aspects of internet security will be subject of additional studies scheduled to appear later in 2007 and in 2008.

1 Introduction

When a network is connected to the Internet, the IT systems and the information processing are exposed to a great number of threats and attacks that don't exist in a disconnected environment. It is therefore necessary to modify the structure of a network to provide for adequate protection.

In the basic architecture proposed by the ISi-LANA study, both using and providing the most popular services Web and E-Mail are supported. The concept can be adapted to the size and complexity of the network infrastructure, the number and types of services to support and the individual security requirements of applications. The study provides a number of variations covering on one hand small and non-critical IT infrastructures and on the other hand supporting high security requirements by supplementary measures or modular extensions.

The underlying principles of the proposed architecture can be summarised as follows:

- *Defence in depth*: There exist several independent layers of defence. There must be no single point of failure that would allow an attacker to gain access to the internal network after overcoming just one line of defence.
- *Separation of functionality*: Independent functions should be implemented independently ("one service – one server"). This holds especially for security-related functionality. This serves to reduce the complexity of the configuration of security components, servers and services, thereby minimising the "contact surface" and the load of the individual components.
- *Minimality*: All components - especially those of the security gateway itself and of the Internet-facing servers – should be configured in a minimal way. Unnecessary software should be deinstalled, unnecessary functionality should be deactivated.
- *Need-to-Know*: System components, applications and services may only disclose such information on the network and its users, that are essential for the functioning and the use of the IT infrastructure. Available information should be protected according to a role and permissions scheme.
- *Whitelisting*: All filter rules should be such that only explicitly allowed packets or connections are allowed and everything else is blocked ("default deny").
- *Currentness*: Operating systems and applications should always be kept up to date. Available patches should be applied as soon as possible.

2 Basic Architecture

The core principle of the basic architecture is controlling the data flow between the Internet and the local network by a three-stage security gateway in "P-A-P" setup, i.e. consisting of an outer **P**acket filter, an **A**pplication level gateway and an inner **P**acket filter. No connection should be initiated from the "outside" with any computer on the internal network. Moreover, the internal network is divided into several security zones separated by packet filters.

For all network zones, private IP address ranges such as 192.168.0.0/16 should be used; the necessary network address translation will be done on the packet filter *PF1*. Ingress / egress filtering and antispoofing rules should be implemented on the perimeter router.

Regarding the connection to the Internet, the two modes of using and providing Internet services have to be distinguished. While in the first case the communication goes straight over the three steps of the P-A-P security gateway, external requests for any services provided have to be dealt with on a server located in a separate demilitarised zone (DMZ) of the security gateway. These servers can revert to other servers located in downstream security zones of the DMZ.

2.1 Using Internet Services

The P-A-P gateway for using Internet services like WWW and E-Mail consists of the outer packet filter *PF1*, the application level gateway (*ALG*) containing proxy servers for protocols such as HTTP and SMTP, and the inner packet filter *PF2*. These two packet filters should support stateful filtering.

The security gateway must not be circumvented. All traffic between the internal network and the Internet has to pass through this gateway, with the gateway using a strict whitelist policy, i.e. allowing only the explicitly permitted protocols. Furthermore, the *ALG* needs filtering capabilities for all supported protocols, notably an E-mail virus (or rather: malware) scanner and a filtering module for the Web proxy. The latter should be configured to filter out most, if not all, Active Content in Web pages. In some cases, JavaScript and other types of Active Content may be required for certain Web sites. Generally, Active Content should only be allowed on a case by case basis for selected sites on a whitelist.

To allow complete control, there can, in general, be no encrypted traffic through the security gateway. Checking this type of traffic requires proxy functionalties on the application level gateway (*ALG*) which allow the decryption and re-encryption of protocols like HTTPS. End-to-end encryption will only be permitted on a case by case basis for selected partners and trustworthy Web sites. While this has some drawbacks, it is a necessary trade-off providing an acceptable security level at the expense of some functionality. As an example, certificate checks for SSL-protected Web sites will not be possible for the end users and will have to be made by the SSL proxy component of the *ALG*. This is, however, the only way to keep malware from directly reaching the internal network through encrypted tunnels.

2.2 Providing Internet Services

In the basic architecture, the Internet-facing services *DNS* and *WWW* are covered. A separate server for the administration and an internal web server (*WWW int*), as well as a web application server (*WWW AS*) capable of serving dynamically-generated pages, are foreseen. The *WWW int* server is available as a frontend for the administration of the web services provided, and for providing dedicated web services for users from the internal network. Requests from the internal network to the *WWW int* server have to pass through the ALG and the servers are separated from the ALG by a packet filter, hence there is a P-A-P gateway between them and the clients on the internal network.

As a backend for the web service, a database server (*WWW DB*) is included in a separate security zone. The communication between this server and the *WWW AS* application server is controlled by a packet filter, limiting the traffic to the necessary protocols, e.g. ODBC, JDBC or a vendor-specific proprietary protocol. If the database server has to be synchronised with an internal database server, this can be implemented over the management network.

The Internet-facing *WWW* and *DNS* servers are located in a forward security zone of the Security Gateway, separated from the Internet only by the packet filter *PF1*. Therefore, they have to be separated from the downstream servers by the packet filter *PF3*. There is no implicit trust relationship between the *WWW* server and the clients in the internal network. If clients from the internal network want to use the „external" Web service, the requests are routed through the P-A-P gateway in the same manner as they would be for an external Web server. That way, an intruder who might have captured the *WWW* server, can not use this server as a starting point for attacking computers in the internal network e.g. by placing malicious JavaScript code in the served pages, because malicious code will be filtered out by the application level gateway component.

2.3 Administration and Monitoring

Administration and monitoring of all servers and security components in the Security Gateway zone are done "out-of-band" over a separate management network.

In the management network, log and monitoring data are collected from the individual devices e.g. via remote syslog or SNMP and are stored on one or several management servers (in Fig. 1, only one server is shown). This allows for central evaluation and correlation of events, giving the administrators the opportunity to quickly detect and react to problems.

In the same way, data from host- or network-based intrusion detection sensors can be collected and processed. If properly implemented, this can give the network early-warning capabilities.

A time server, connected to a high precision time source such as GPS or DCF-77 is placed in the inner zone of the management network. This

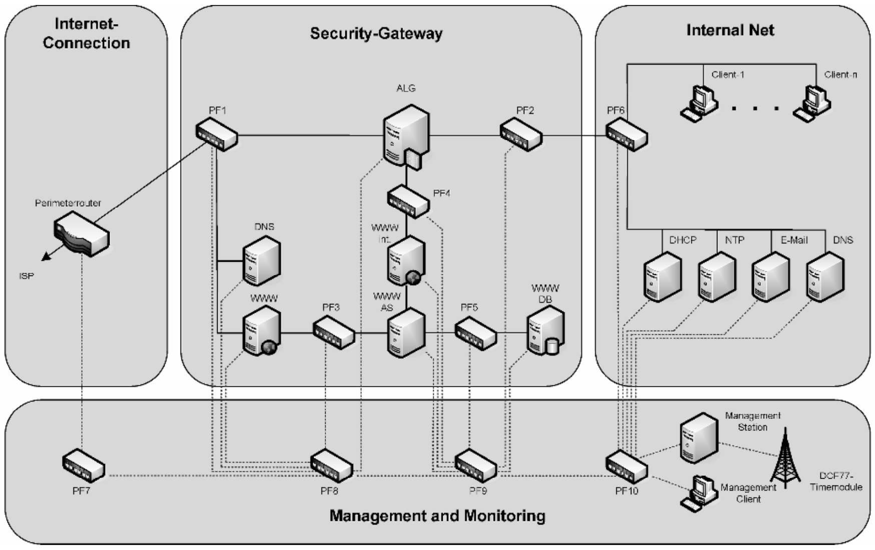


Fig. 1. Basic architecture including management and monitoring network.

server is used to synchronise the system clocks of all components of the management and security gateway zones, as well as for the internal NTP server, which provides a network time source for the computers on the intenal network.

The management network is divided in several zones by the packet filters *PF7 – PF10*.

2.4 Implementation and Operations

As the network architecture alone can not guarantee the security of the network, in the ISi-LANA study the basic architecture is complemented by extensive advice concerning implementation and network operations. Included are

- basic requirements for all network and server components,
- more specific requirements for switches, packet filters, application level gateway components and servers,
- general configuration advice for the network devices, application level gateway components and servers,

as well as some points concerning operations. For more details concerning procedures and security management, the reader is referred to standards such as the BSI Grundschutz Manual (GSHB).

3 Discussion

3.1 Overall Structure

The overall structure was developed during extensive discussions. Requirements from various sources such as the BSI Grundschutz Manual (GSHB) and the E-Government Manual (EGov) were collected and combined. The resulting structure derives naturally from the requirements and the basic principles mentioned above.

The (ISi-LANA 2007) study provides an extensive list of well-known network-related threats on the lower layers of the OSI reference model. Examples include

- MAC- or ARP-spoofing,
- attacks on the network infrastructure through inter-switch protocols (e.g. STP or VLAN-related protocols) or routing protocols,
- attacks via basic properties of the IP protocol suite (e.g. related to ICMP, fragmentation or broadcast mechanisms),
- TCP- or UDP-related attacks (such as TCP connection hijacking), and
- attacks on DNS and other "service" protocols.

It was assumed that the network has normal protection requirements. The network structure should provide an adequate level of protection against these threats, while keeping complexity and cost at a reasonable level.

When making trade-offs between security and functionality, security was given precedence over functionality or cost in most cases. In real life, things may not always work out that way. For organisations with a higher risk acceptance, or budget constraints, it is possible to simplify the structure in several places. In (ISi-LANA 2007), a number of variants are discussed, describing the possible benefits and the associated risks. On the other hand, variants providing better protection for networks with higher protection requirements are also described.

Two examples can serve to illustrate the way the network architecture was developed:

- Out-of-band management is a result of the following requirements: firstly, no connections should be made from outside the security gateway to the inside network. Secondly, on one hand no encrypted tunnels should be permitted, but on the other hand insecure and outdated protocols like telnet and SNMPv1 should not be used on the "main" network because the data transported over these protocols can be of rather sensitive nature. The separate management network solves this apparent conflict in a natural way.

- The place of the network address translation was chosen to be the packet filter *PF1* for simplicity. Translation further downstream is not necessary due to the requirement that all outbound requests and transmissions should pass through the application level gateway *ALG*, which will terminate the connection coming from the internal network and initiate a new connection on its own external interface. Therefore, the address of the *ALG* is the only address that will show up on *PF1*. Nevertheless, the ALG should not get a "public" IP address because the NAT will protect its TCP/IP stack from direct attacks.

3.2 Structure of the Security Gateway

The security gateway is structured in a way that implements different security zones even inside the security gateway itself. Some reasons for this were already discussed above.

The separate internal web server *WWW int* helps to secure the administration interface of the web application server *WWW AS*. Often, web applications have their adminstration and "members only" sections reachable more or less directly from the front page. That way, all that is between an attacker and the administration interface is a password and a piece of software that often is not implemented very securely. By separating the management and the internal web service from the outward facing server (assuming that the application server and web content management system support this), this potential problem is eliminated.

Routing all requests even from the clients on the internal network through the application level gateway provides an additional line of defence against attacks on the application layer.

Using separate packet filters *PF3* - *PF5* instead of just putting all servers on different interfaces of a larger packet filter was chosen to keep the complexity of the filtering rules down. Using only one packet filter may be possible, but as the complexity of the filtering rules grows very quickly with the size of the communication matrix, the administration of such a central hub device would be quite tricky and the risk for configuration errors would be high.

3.3 Structure of the Management Network

While using four packet filters in the Management Network zone may seem like overkill at first, it is a necessary precaution in order to implement the defence in depth principle. If, for example, only *PF7* was used as a single packet filter, this would constitute a single point of failure that, if captured, would allow an attacker to completely circumvent the security gateway and reach the internal network in only one step. In the proposed setup, an attacker will still find himself completely outside the security gateway even after capturing *PF7* and will need to compromise several other security devices before finally reaching the internal network.

Unlike the “main” packet filters *PF1* and *PF2* in the security gateway zone, *PF7 – PF10* need not necessarily support stateful filtering. Also, the bandwith requirements of the management network will usually be lower than those of the main internet connection. Therefore the packet filters in the management network can be built with cheaper hardware.

3.4 Structure of the Internal Network

The proposed structure of the internal network is not overly sophisticated: the servers are separated from the clients by a packet filter, which should support stateful filtering. Network security inside the internal network consists mainly of configuration options on the switches to limit the impact of attacks on the lower layers of the ISO model, such as ARP or ICMP spoofing.

One more recommendation should be noted at this point: using the VLAN functionality of switches should not be considered a security measure. VLANs can be used to keep broadcast domains small, but they should not be used to separate security zones.

(ISi-LANA 2007) also discusses variants for higher security requirements, such as segmentation of the client network or using a more sophisticated security gateway to protect the server zone.

4 Conclusions

The basic architecture proposed by ISi-LANA provides a good level of protection against threats on the network level and also limits the options of an inside attacker. Due to its modular and extensible design, the architecture can be adapted to different requirements and thus provide a starting point for the design of secure networks in most scenarios.

The study also presents variants for organisations with higher security requirements.

It has to be noted that the implementation of the network structure needs to be complemented by appropriate security measures on the clients in the internal network, ranging from secure configuration to an up to date malware protection. These and other aspects of the security of networks connected to the Internet will be covered in separate modules of the ISi series. The upcoming modules “ISi-Mail” and “ISi-Web” will provide a more in-depth coverage of the most used services in internet-connected networks.

Author Index

Subject Index