Efficient Prevention of Credit Card Leakage from Enterprise Networks

Matthew Hall[1], Reinoud Koornstra[2], and Miranda Mowbray[3]

[1] No Institutional Affiliation
mhall@mhcomputing.net
[2] HP Networking, USA
koornstra@hp.com
[3] HP Labs, UK
miranda.mowbray@hp.com

Abstract. We have developed a new approach to the problem of preventing the leakage of credit card numbers in traffic on a large enterprise network. In contrast to a previously-used method, it has higher throughput, and it can be partly implemented in hardware without any additional libraries.

Keywords: cloud security, privacy, data leak prevention.

1 Previous Approaches to Preventing Credit Card Leakage

The danger of credit card numbers leaking from enterprise networks onto the public Internet is exacerbated both by the rise of targeted phishing attacks, and by the increasing use of cloud computing and consequent increase in data traffic between enterprises and the public cloud.

Several companies (for example Symantec, Websense, Vericept, Mimecast and Code Green networks), offer products and services that examine the data layer of a packet on an enterprise network and determine whether it contains credit card numbers, so as to prevent these from being leaked.

These products and services use one of two approaches. The first approach is to store digital fingerprints of a set of card numbers and check the data for exact matches to these digital fingerprints. This however can only detect credit card numbers whose fingerprints are in the stored set.

The second approach, which can detect any card numbers, begins by performing a first pass on the packet data to identify candidate numbers that fit a regular expression for potential card numbers. For example, all American Express (Amex) card numbers are 15-digit numbers beginning with 34 or 37. Then a second pass is performed to determine if any of the candidate numbers satisfy a check called the Luhn check. All valid credit card numbers satisfy this check, although the converse does not hold.

One way to carry out the Luhn check on a number n is to double every alternate digit in the number including the penultimate digit, sum the digits of the resulting numbers, set $L(n)$ to the remainder mod 10 of this sum, and check whether $L(n) = 0$. For example, 932152 passes the Luhn check because the sum of the digits of the numbers 18 3 4 1 10 2 is 20, but 93215 does not because the sum of the digits of the numbers 9 6 2 2 5 is 24, which is not divisible by 10.

The candidate numbers that are found to pass the Luhn check are sent to leakage inspectors for Amex, VISA etc. that have full information about the set of valid numbers issued by the provider. If a valid credit card number is identified, further transmission of the packet containing this number may be blocked.

Unfortunately, the regular-expression pass is an expensive operation in terms of resource requirements. Our experience is that for the high packet volumes on modern enterprise networks, it is infeasible to run the regex pass on all packets. This pass might be speeded up by implementing it in hardware: however this would require the regex library, which has considerable size, to be stored in the hardware.

2 Our Approach

We have developed and prototyped a new approach to this problem. Instead of first performing a regex pass and then a Luhn check pass, we first use a novel high-speed streaming Luhn algorithm which identifies, in a single pass, all 14, 15 or 16-digit numerical substrings of the data packet that pass the Luhn check. (We have applied for a US patent for this streaming algorithm, application number PCT/US2011/022709). Lightweight custom string check functions, in software, are then run on the set of numbers that are reported by the algorithm as passing the Luhn check. These string checks are designed so that the Luhn check algorithm and string checks between them carry out exactly the same set of checks as the first two passes of the approach that uses regular expressions. Any candidate card numbers passing these checks are forwarded to leakage inspectors, as before.

In a benchmarking experiment (details of which are omitted from this extended abstract for lack of space), a software implementation using our approach achieved more than 4.7 times the throughput of an implementation of the approach using a regex pass. If even faster throughput is necessary to process high traffic volumes, our Luhn check algorithm is simple and easy to implement in hardware, without the need for additional libraries.

The pseudocode for the algorithm is below. The notation `sd(a,b)` is shorthand for the string with entries `d[a],d[a+1], ... d[b]`, where `a`, `b` $\in \mathbb{Z}$ and $0 \leq$ `a` $\leq$ `b`. The vector `d` stores the sequence of digits received from the stream since the beginning or the last non-digit character, and `i` records this sequence's length. When a new digit is read in from the string, `i` is updated and the variable `x[i]` is set such that `x[i]` is equivalent mod 10 to $L($`sd(1,i)`$)$. Then the algorithm determines whether the substrings of length 13, 14 or 15 ending at this new digit pass the Luhn check.

To determine this, the algorithm uses the fact that if $s1$, $s2$ are digital strings and $s2$ is of even length, and $s1 \cdot s2$ is the concatenation of $s1$ with

$s2$, it follows from the definition of L that $L(s2) = (L(s1 \cdot s2) - L(s1))\%10$. If $i > 13$, putting $s1$=`sd(1,i-14)`, $s2$=`sd(i-13,i)` in this equation implies that `sd(i-13,i)` passes the Luhn check iff $(L($`sd(1,i)`$)-L($`sd(1,i-14)`$)\%10 = 0$, which is equivalent to $($`x[i]`$-$`x[i-14]`$)\%10 = 0$. The checks for `sd(i-14,i)` and `sd(i-15,i)` can be derived similarly by setting $s1$=`d[i-14]`, $s2$=`sd(i-13,i)` and $s1$=`sd(1,i-16)`, $s2$=`sd(i-15,i)` respectively.

```
Start by setting i=0, d[0]=0, x[0]=0.
While there are more entries in the string, repeat the following:
   Get the next entry, and set e to it
   if e is other than a base-10 digit
      set i = 0
   if e is a base-10 digit
      increase i by 1
      set d[i] = e
      if i == 1 set x[1] = e
      if i > 1
         set x[i] = d[i] + 2d[i-1] + x[i-2]
         if d[i-1] > 4 increase x[i] by 1
      if i > 13
         set c = (x[i] - x[i-14]) % 10
         if c == 0  report sd(i-13,i) as passing the check
      if i > 14
         add d[i-14] to c
         if c % 10 == 0  report sd(i-14,i) as passing the check
      if i > 15 and (x[i] - x[i-16]) % 10 == 0
         report sd(i-15,i) as passing the check
```

This algorithm could be further refined. For instance, lookup tables can further reduce computation requirements, and memory requirements can be reduced by over-writing all but the 17 most recent elements of the vectors, since only these are used. The number of strings processed by the string check can be reduced at the expense of slightly more computation during the Luhn check pass, by modifying the Luhn check algorithm to only report numbers beginning with 3, 4, 5 or 6.

Using a streaming algorithm in place of a regex check might also speed up the detection of other types of personal data, for example IBAN numbers or numbers in some national ID schemes.

There is increasing use of protocols such as SSL which transmit data in encrypted form. This protects data in transit, but leaves open the possibility that personal data may be transmitted by mistake and misused after it has been decrypted by the recipient. Companies such as Symantec, Code Green Networks and Trend Micro offer products that can intercept data before transmission (they are known as Endpoint DLP products). If used in combination with some interception means, our method could be used to inspect data before transmission, and block the transmission where necessary.