Reprinted from the Proceedings of the 1990 SIGSMALL/PC Symposium on Small Systems published by ACM Press.

Adequacy of Checksum Algorithms for Computer Virus Detection

Doug Varney
Kansas State University

1.0 Introduction

Checksums, long used for random error detection in communications, is now being employed to detect changes for integrity purposes. For example, checksums are being used for the detection of computer viruses [POZ86]. The checksum algorithms for detecting random errors are not sufficient against an entity that wishes to "fool" the checksum mechanism. This entity wants to be able to insert a forgery in place of the original data such that an unsuspecting user does not realize the forgery has occurred. This paper describes checksum algorithms and features of checksum algorithms to deter this type of forgery.

2.0 Checksums

A checksum, or digital signature, is any fixed length block functionally dependent on every bit of the message, so that different messages will have different checksums with a high probability [DEN82]. A checksum is generated by a checksum algorithm. Probably the most used checksum algorithm family is that of Cyclical Redundancy Codes (CRC) which are used extensively in communication. General checksum algorithms, such as the CRCs must produce checksums with the following features:

1) Even mapping - the probability of producing any checksum is approximately equal to generating any other checksum. Given a checksum of n bits this probability is equal to 2**(-n).

2) Permutation Sensitivity - different ordering of data items within a data set produce different checksums. I.e. the checksum of ABC produces a different checksum than ACB.

3) Overdeterminism - the checksum is functionally dependent on every bit of the data being checksummed. If this were not the case then errors in the bits of the data which did not affect the checksum would be undetected.

These features are sufficient for random errors or random bursts of errors, but are not sufficient for an active attacker who introduces calculated nonrandom errors.

3.0 Attacks Against Checksums

An attacker defeats a checksum algorithm, f(s,d)=c where s is the seed, d is the data, and c is the checksum by determining a set of data, D', to insert in place of the original data, D, such that the checksum of D and D' are equal: f(s,D) = f(s,D') There are three general attacks against checksums: brute force, birthday, and trap door.

Brute Force Attack

A brute force attack involves generating many different sets of data and corresponding checksums until a match with the checksum of the original data is found. Typically only a small portion of the original data is changed in order to maintain the functionality of the original data. The last block of the changed data is mutated until the proper checksum is found. With even mapping, the probability of generating a checksum, with any single mutation, that matches the original data is 2**(-n). It is necessary to generate ln(2) * 2**n mutations before there is a 50% probability of a successful attack. For example, for a 16 bit checksum approximately 40,000 mutations are necessary before there is a 50% chance of success.

Birthday Attack

The birthday attack is a forgery accomplished by the originator of the data. This attack involves generating and storing many variations of a original set of data and corresponding checksum and many variations of the set of data and corresponding checksums to be inserted. Since any pair of original data and forged data provides a successful forgery the number of variations needed to be generated is greatly reduced. A birthday attack will succeed by producing a forgery 50% of the time after 2**(n/2) checksums. The standard birthday attack can only be accomplished by an attacker which has access to the original data before it is originally checksummed.

Trap Door Attack
A trap door is essential a short cut to determine a set of data to insert as a forgery. If an algorithm exists to determine a set of data that generates the same checksum and requires less computational effort than a brute force attack, then a trap door exists.

4.0 Features for protection against attackers.

The two basic features to protect against attackers are the length of the checksum and the noninvertablity of the checksum algorithm.

Length of the Checksum.
The length of the checksum is its the number of bits in the checksum. The checksum should be of sufficient length such that the cost of generating enough variations in a brute force attack is unacceptably high to the attacker and, if necessary, provide protection against birthday attacks. A 32 bit length checksum provides adequate protection from brute force attacks while 128 bit length checksum is necessary to provide protection from birthday attacks [JUE86].

Noninvertable Algorithm
A noninvertable checksum algorithm is a function that cannot be inverted. Thus given a checksum algorithm, f(s,d)=c there does not exist either a function g(s,c)=d or h(c,d)=s. Either of these functions lead to trap doors.

If there exists g(s,c)=d then the attacker can insert a additional data segment knowing what the checksum should be, calculate the checksum up to that point giving the seed, then calculate the appropriate "filler" that will make the checksums match.

If there exists h(c,d)=s the attacker inserts the desired data segment followed by two filler data segments. The first filler data segment is mutated to give a series of checksums using f(s,d)=c. The second filler data segment is mutated and a series of seeds generated using h(c,d)=s. the checksums resulting from the first filler is compared with the seeds calculated from the second filler. A successful pair of filler blocks will be found on average in 2**(n/2) generated checksums. This is a variation of the birthday attack.

Either of these inversions lead to trap doors. Trap doors are the most serious threat because of the much reduced effort to generate forgeries. Unfortunately there is no test to show that a trap door does not exist for a checksum algorithm.

5.0 Construction Techniques

The construction of checksum algorithms that provide noninvertablility is similar to those used in cryptography. The three methods used are substitution, transposition and feedback. Substitution involves replacing one block of original data with a corresponding block from a ciphertext alphabet. Transposition is the rearranging of blocks of data according to some scheme. Feedback is the use of previous information in the computation of a ciphertext block. Nonlinear feedback provides permutation sensitivity and should be employed. When constructing checksum algorithms the following functions are useful: exclusive-or, raising to a power, modular arithmetic and multiple equations using the same data.

Along with cryptography, a fruitful source for parts of checksum algorithms are random number algorithms. Though generally not suitable for direct use, random number algorithms have many of the same features that are desirable in checksum algorithms. A good source for established forms of checksum algorithms, though either directly concerned with preventing Birthday Attacks or checksums that are encrypted with the original set of data, is Jueneman work [JUE83] [JUE86].

6.0 Tests for Checksum Algorithms

It is necessary to test 1) that the checksums generated by a checksum algorithm provide an even mapping and 2) that the checksum algorithm is noninvertable. Statistical methods are employed for testing for even mapping. Three statistical tests to test the even mapping property are chi-square, collision, and binomial.

The chi-square test compares the expected distribution (even mapping) with the checksums generated from executable programs as data. Since the number of programs needed to have a statistical significant number at each possible checksum is very large (for adequate checksum lengths) the output range of possible checksums is divided into equal groups. A checksum for any program should have the same probability for each of the groups due to even mapping. Thus a chi-square statistic for all the checksums can be generated and compared to tell if statistically significant.

The collision test is based on the fact that even though there are many possible checksums while the number of checksum actually generated is relatively small, we can expect some of the programs to have the same checksum. Using the formulas in Knuth [KNU82] it is possible to tell if this number of "collisions" is in the expected range. The collision test is probably only applicable to checksums of length 32 or less, otherwise the number of checksums generated will have to be very large in order to find collisions.

The binomial test is a check to see if each bit of a generated checksum has equal probability of being a zero or one. To test this bit wise probability the number of one (or zero) bits is calculated for each checksum. The resulting distribution should be a binomial distribution. The amount the generated distribution differs from a binomial distribution can be tested for significance using a chi-square test.

There is a difficulty in finding enough programs the generate checksums for in order be able to test statistical significance. Random numbers can be used to simulate programs for the purposes of statistical significance .

The testing of a program for noninvertability is more difficult than testing for even mapping because no standard method exists for testing for noninvertability. Each algorithm needs to be scrutinized for possible noninvertability. Noninvertability must be considered in the case of existence of a general inversion algorithm, where any (checksum, seed, data) can be inverted, and specific inversion algorithms where only a specific combination of (checksum, seed, data) can be inverted. Cohen provides an example of a specific inversion of the original form of his cryptographic checksum [COH86] [COH88].

In the course of our investigation we developed a checksum algorithm that satisfied these tests and was efficient enough for use on microcomputers.

7.0 Conclusions

In future we can expect viruses to actively attack virus detection systems which use checksums to determine changes. A virus attempts to determine a similar program, which it has infected, that has the same checksum as the original uninfected program. The virus can determine the infected program either by using a brute force method or a trapdoor. In order to defeat these attacks the checksum algorithm must have the features of a general checksum algorithms (even mapping, permutation sensitivity, and overdeterminism), be long enough to defeat a brute force attack, and be noninvertable. The checksum algorithm must be tested for suitability to provide even mapping and noninvertability. The even mapping can be accomplished by statistical tests, but there is no standard method of testing noninvertability.

As viruses become "smarter", checksum algorithms must increase in complexity in order to provide protection. This paper outlines the general methods to properly accomplish this complexity.

References

[COH86] Cohen, F., A Cryptographic Checksum for Integrity Protection, Computers and Security 6 (1987), 505-510.

[COH88] Cohen, F., On the Implications of Computer Viruses and Methods of Defense, Computers and Security 7 (1988), 167-184.

[DEN82] Denning, D., Cryptography and Data Security, Addison-Wesley Publishing Company, Reading, MA, 1982.

[JUE83] Jueneman, R. R., Matyas, S. M., Meyer, C. H., Message Authentication with Manipulation Detection Codes, IEEE Conference on Security and Privacy 1983, 33-54.

[JUE86] Jueneman, R. R., A High Speed Manipulation Detection Code, CRYPTO 86 (1986), Santa Barbara, CA, 327-346.

[KNU81] Knuth, D. E., The Art of Computer Programming, Volume 2, Seminumerical Algorithms, Addison-Wesley Publishing Company, Reading, MA, 1981.

[POZ86] Pozzo, M. M., Gray, T. E., An Approach to Containing Computer Viruses, Computers and Security 6 (4), August 1987, 321-331.