

Worm Detection and Prevention: Concept, Approach, and Experience

14 August 2002

Todd Heberlein
Net Squared, Inc.
todd@NetSQ.com

The following material is part of our Network Radar final reports. We detail a detection technology we call thumbprinting. This is followed by a potential use of the technology in an operational environment to stop fast spreading attacks like worms. Finally, we present some experience with the approach during the Love Bug attack at Rome Labs.

1 Introduction

Thumbprints were first proposed in 1992 in a paper titled “Internetwork Security Monitor: An Intrusion-Detection System for Large-Scale Networks”¹. The primary purpose for thumbprints as proposed in that original paper as well as a follow up paper describing an initial prototype² was to track users hopping across the network via interactive login services such as telnet and rlogin. Because of trust relationships between hosts (.rhosts and hosts.equiv), default passwords, weak passwords, and shared passwords across multiple systems, attackers could penetrate deeply into the Internet with these interactive login services.

Unfortunately, by the late 1990s when we started deploying a thumbprint-based sensor (Network Radar) in an operational environment, most sites which were concerned with security had largely disabled the use of the interactive plaintext telnet and rlogin services. When interactive login was necessary, these sites used the Secure Shell protocol, SSH. Because our thumbprint approach analyzed the content of the interactive login sessions (e.g., keystrokes and the data displayed on the user’s screen), and because SSH encrypts this content, thumbprint analysis was no longer as effective as it would have been in the early 1990s.

In 1999 and 2000 we investigated other uses of thumbprints, and we applied thumbprints to entire network sessions to look for the same content crossing multiple network connections. The purpose of this approach is to detect new attacks launched against multiple servers or worms spreading across a network. The thumbprint can even be used as a signature, and when deployed in interdiction devices (e.g., firewalls) thumbprints could potentially be used to stop a newly detected attack.

¹ L.T. Heberlein, B. Mukherjee, K.N. Levitt., “Internetwork Security Monitor: An Intrusion-Detection System for Large-Scale Networks,” Proc. 15th National Computer Security Conference, pp. 262-271, Oct. 1992.

² S. Staniford-Chen, and L.T. Heberlein, “Holding Intruders Accountable on the Internet”. Proceedings of the 1995 IEEE Symposium on Security and Privacy, Oakland, CA, 8-10 May 1995, pp. 39-49.

In section 2 we present what a thumbprint is. In section 3 we compare thumbprints with the more traditional signature-based approaches. We also discuss what thumbprints can do that classical signatures cannot: detect new attacks by looking at repeated activity. In section 4 we present a simple operational model using these concepts to stop a fast moving worm. Finally, in section 5 we wrap-up our tour of thumbprints with an actual example of using this approach to track the Love Bug worm sweeping through the Air Force’s Rome Labs.

2 Thumbprint Concept

A thumbprint is simply a small numerical representation of some content. It is related to the more familiar hash functions and checksums. Figure 1 shows the basic approach. The original content, “The quick brown fox jumped over the lazy dog.” is sent through a hash function that generates a number. The original content consisted of 360 bits (8 bits per character times 45 characters) and the result in a single 32-bit number (a typical unsigned integer on most computers).

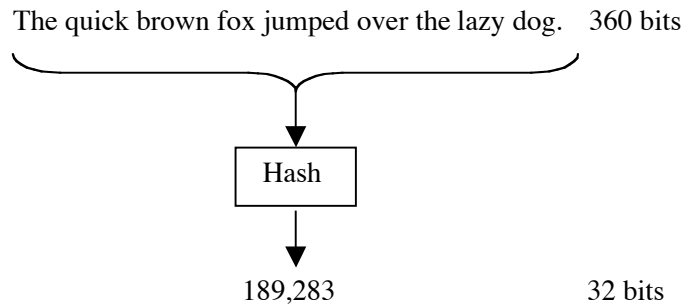


Figure 1: Numerical Representation

This number can serve as type of compact representation of the original content. For example, suppose a document is processed by this technique. A hash number is computed for each sentence in the document, and then we store all the numbers representing all the sentences in a hash table. Later, if a user provides us with a sample sentence and asks us if it is in the document, we can use the following algorithm to very quickly determine the answer. First, compute the hash value of the sample sentence. Second, look in the hash table to see if that number exists in the table. If it is not in the table, then the sample sentence is not in the document and we are done. Third, if we do find a matching hash value in the table, then we examine the sentence (or sentences) in the original document that created the hash value and determine if it matches our sample sentence.

This is roughly the way we use thumbprints to identify duplicate content crossing the network. We compute a thumbprint of each network connection observed (essentially the sample sentence provided by the user in the above example), and then we look in our hash table to determine if we have potentially seen the same content in another connection.

Unfortunately, traditional hash functions do not work well for our problem area. Most hash functions are designed to produce a completely different hash number even if the content only varies by a single byte. For example, in Figure 2 we modified the original sentence by making the word “dog” plural, “dogs.” This single change produces a completely different hash number. In fact, in traditional hashing functions, looking at just the resulting numbers would not indicate that sentences **A** and **B** were very similar, and sentence **C** was completely different than both of these.

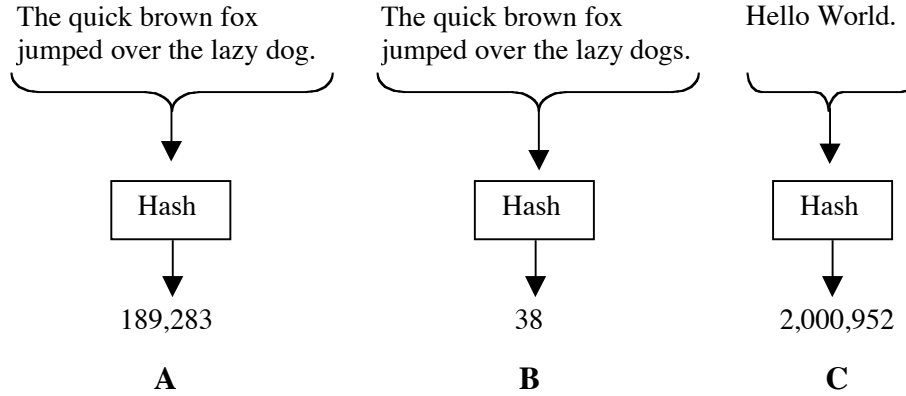


Figure 2: Traditional Checksums

The reason that this traditional approach does not work for us is that the same attack, as observed from our network sensor, might appear slightly differently from one instance to another. For example, in the case of email worms, the content of the email connection would include the sender and recipient’s email addresses, and these would vary each time the worm attempts to infect a new machine. Therefore, we need a hashing function that preserves the similarity. Our hash numbers for similar content should be similar. Blain Burnham described this approach as a “robust checksum.”

Figure 3 demonstrates how our thumbprint hashing function might treat the three sample sentences from the earlier example. Now the second sentence, which varies from the original sentence by the addition of the single letter ‘s’, has a hash value that is only slightly different from that of the original sentence. Now when the user presents a sample sentence to our hypothetical document analysis system, we can quickly determine whether the exact sentence, *or one very close to the sample sentence*, exists in the original document.

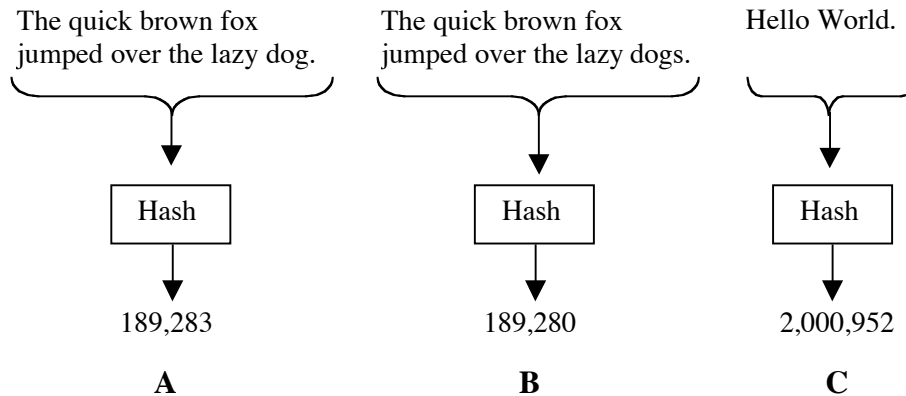


Figure 3: Thumbprint Checksums

These are the basic concepts of a thumbprint: condense a potentially large amount of content to a single number, and similar content should generate similar numbers. There can also be variations. For example, we might condense the content to two numbers instead of one (think of a two dimensional vector). Once again, for similar content, each pair of numbers should be similar. By using two numbers, however, we reduce the probability of a false positive, where two dissimilar content samples would have both pairs of numbers similar. One approach that satisfies all of these needs, and the one we use to in our implementation, is a popular technique in

multivariate statistical analysis called principal component analysis (PCA). Other algorithms could potentially be used.

3 Comparisons to Traditional Detection Techniques

The thumbprint approach just described is simply a mechanism that can be used for quickly comparing content. If an exact match is needed, a traditional checksum can be used. If a more robust match is required, our PCA approach can be used. In this section we compare this approach with traditional content-based detection approaches used in intrusion detection systems. We also discuss how this approach can be used in way that traditional signature-based detection cannot be: looking for original attacks on the network.

Most traditional network-based intrusion detection systems use some type of key-word detection algorithm. The early UC Davis Network Security Monitor (NSM), Air Force's ASIM monitor, LLNL's NID, and DISA's JIDS intrusion detection systems all used the Knuth-Morris-Pratt pattern matcher. Snort uses a Boyer-Moore pattern matcher. Early versions of WheelGroup's NetRanger used the default UNIX regular expression matcher, which in turn used a basic non-deterministic, backtracking state machine.

All these approaches essentially do the same thing: they all look for strings in a network data stream. The strings are hand-generated by analysts. And the systems look for each string one at a time. If you double the number of strings you are searching for, you double the analysis time that must be spent on each packet.

Our thumbprint-based approach, on the other hand, calculates a vector (perhaps just a single number (e.g., a single hash value) representing a one-dimensional vector or maybe several numbers for a multi-dimensional vector). To compare a vector of a new connection against known attacks, we simply compute the distance between our new vector and vectors of known attacks. However, because the vectors are numerical in nature, we can hash the vectors into buckets, so the new vector can be compared against all known attack vectors (perhaps thousands) simultaneously by simply looking into the appropriate hash bucket. Thus, when trying to solve the traditional problem of looking for known patterns, this approach scales to larger signature sets much better than the traditional signature algorithms.

The thumbprint approach can also do something traditional signature-based systems cannot: detect new attacks automatically.

In a traditional operational setting, somehow a new attack is detected. Since most signature-based systems primarily detect what they already know about, new attacks are often "detected" through some other mechanism than the intrusion detection system. Perhaps an attack tool is posted to BugTraq. Perhaps a site captures the new attack with a honey pot. However the new attack is detected, the appropriate evidence is shipped to the analysts (e.g., AFIWC). An analyst carefully examines the attack and then hand-generates a string or pattern that will detect the new attack.

Our approach, on the other hand, can detect the new attack if it is repeated several times, such as when an attack is part of a worm or used against multiple servers (e.g., the attack is sweeping a site). Thus the new attack can be detected in theory on just the second time it passes our sensor. Furthermore, since the vector *is* the signature, the signature is *automatically* created.

In the traditional operational setting, because the attack must be detected "by other means", and the signature must be generated by hand by a human analyst, a good turn around time from first attack to a signature ready for deployment is a day; at best a several hour turn around time can be expected. This is far too slow to affect a fast spreading worm.

Our approach, on the other hand, may have a signature ready to go within a second of the first observed attack. Thus our approach can be used to interdict fast moving attacks.

4 Example Operational Model

Figure 4 shows an example scenario of how this thumbprint-based approach, combined with cluster detection just described, can be used to quickly stop a new worm from spreading. The Air Force network in the upper left corner is attacked multiple times, perhaps by an email worm. Our sensor detects the attack by identifying the cluster of content vectors (1). The vector *is* the signature, and it is reported to a central distribution center (2). The distribution center pushes the signature to interdiction devices such as email gateways and firewalls (3). An email gateway at a university detects the email worm, and places the offending email into a queue to be processed or deleted by a system administrator at a later time (4). Because the detection is automatic, the signature generation is automatic, and the distribution the signature is automatic, the interdiction devices around the globe can interrupt the spread of the worm in minutes from the attack's initial detection at an Air Force network.

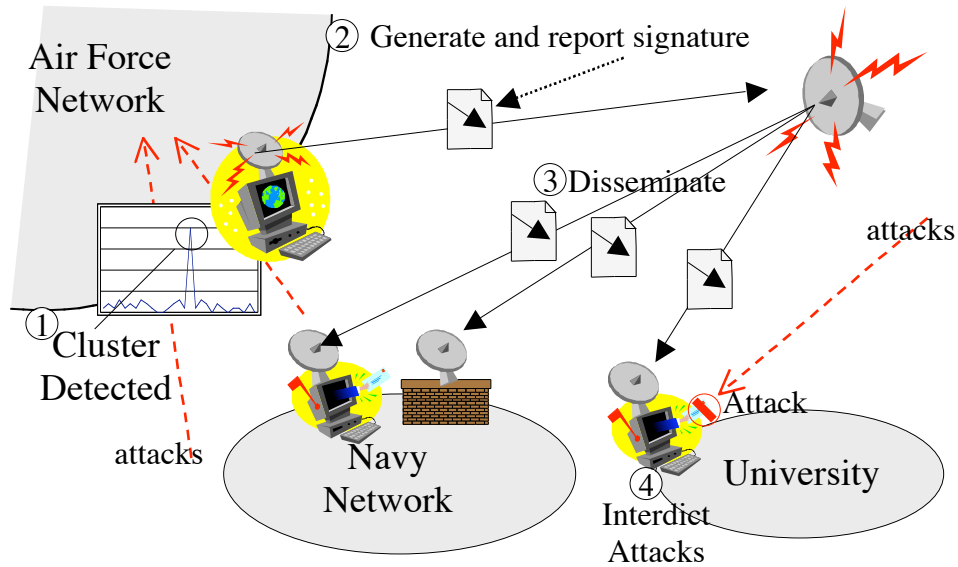


Figure 4: Operational Model

5 Example of Detection in Action

Figure 4 shows a hypothetical scenario. However, Network Radar does thumbprint network sessions (as well as interactive logins), and we have operational experience demonstrating that the approach can work. Network Radar was running at the Air Force's Rome Labs on May 4th, 2000 when the ILOVEYOU virus (a.k.a., the Love Bug worm) hit the Internet. We harvested the data collected by the monitor and generated an animated movie that shows the cluster spikes as the worm moved through Rome Labs. Figure 5 shows several selected frames from the movie.

Since ILOVEYOU was an email-based worm, for several reasons the best place to apply our vector-based cluster detection technology is in an email gateway (as opposed to a network sensor). First, a single sendmail connection can and often does transmit multiple email messages over the same connection. However, from our network perspective we treat the entire connection as a single transaction. Second, as email worms travel from user to user, header information is

Worm Detection

constantly changing, so from a network perspective that cannot distinguish header, body, and attachments in an email message, the representative vector will vary slightly as the headers vary. We could potentially address this with Network Radar since it can easily build a sendmail parser, but we are unlikely to get firewall vendors that will need to interdict the attack later to do the same thing. Third, email systems encode attachments using several different algorithms, so from the network perspective the same worm will look different depending on the encoding scheme used by the email client. An email gateway can decode the attachment and calculate the vector on the original data.

For all these reasons, email-based worms should be detected (and interdicted) via email gateways as opposed to a network monitor that we used. However, despite this, our network monitor version, a first generation prototype at that, was clearly able to identify the spike from the attack.

In each graph in Figure 5, the horizontal axis shows the value a network connection was mapped to (i.e., the hash number described earlier); that is, we were only using a one-dimensional vector to represent each connection. A small circle is placed on this axis to indicate the score our system assigned to a copy of the worm that was sent to us (via our DARPA security mailing list, no less). The vertical axis represents the rates at which connections were observed matching a given vector score. The maximum value on our graph is 500 connections per hour for a given vector value.

Figure 5's panel A shows the initial indications of the virus. We see a small cluster, or spike, at the circle matching the score we received, and we also see a second cluster towards the left. Because the two clusters tracked each other so closely, we believe the second cluster was the same worm where the email attachment was encoded with a different algorithm. Unfortunately (or fortunately depending on your perspective) we only received a single copy of the worm, so we could not verify the second encoding hypothesis. The spread of the worm continues to grow through panels B, C, and D, until it reaches its maximum peak in panel E at 8:38am. Following the peak the virus rate declines in F and reaches almost the level of background noise in panel G at 10:45am. Finally the virus shows another increase beginning around 11:00am and a second smaller peak in panel H at 11:24am. This second period of growth corresponds to the eight o'clock hour on the west coast – just when people are showing up to work and opening their email there.

This real-world example demonstrates two important points. First the worm spread very quickly, reaching a peak about 35 minutes after initial penetration and almost burning itself out in less than three hours. The classic intrusion detection and interdiction systems cannot compete with this time scale. Second, the basic approach of vector-based cluster detection does work, even in this less than optimal situation.

Worm Detection

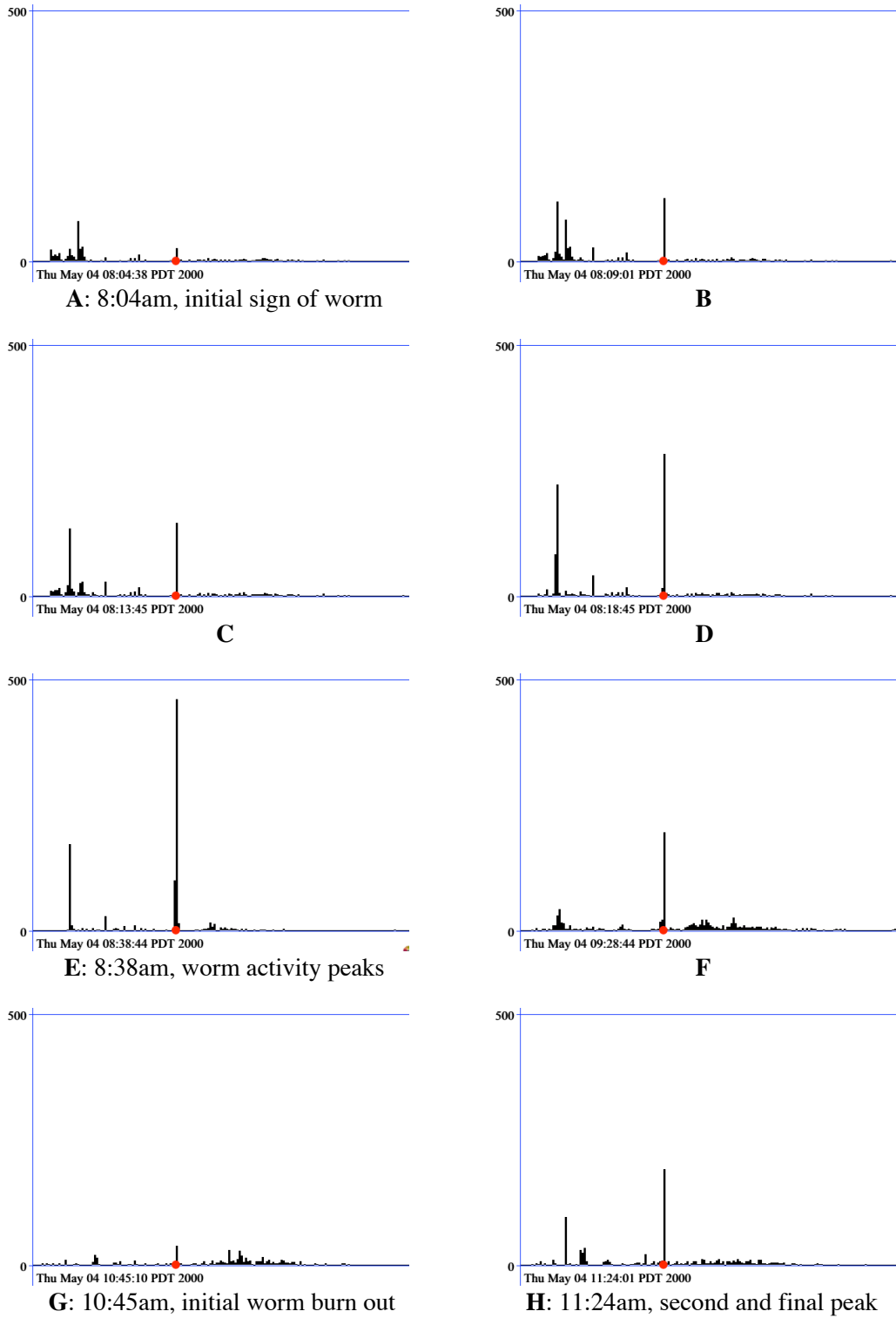


Figure 5: ILOVEYOU Worm Strikes Rome Lab