Security Vulnerabilities and Solutions for Packet Sampling

Sharon Goldberg and Jennifer Rexford
Princeton University, Princeton, NJ, USA 08544
{goldbe, jrex}@princeton.edu

This is the full paper from May 24, 2007.

***Abstract*— Packet sampling supports a range of Internet measurement applications including characterizing the spatial flow of traffic through a network for traffic engineering purposes, identifying the flows utilizing a link for billing purposes or for intrusion detection, and monitoring end-to-end data-path quality. However, packet-sampling mechanisms must be robust to adversarial hosts that craft packet streams that are disproportionately selected by a packet sampler. For example, a botnet flooding a network with packets in a denial-of-service attack, or a greedy customer trying to avoid being billed for network utilization, each have a strong incentive to craft packet streams that evade selection by the packet sampler.**

In this paper, we focus on securing the passive packet sampling mechanisms recommended by PSAMP (the IETF Packet Sampling working group [1]) against adversarial hosts. We show that (1) some of the packet sampling techniques suggested in current drafts of the PSAMP charter have security vulnerabilities, (2) secure uncoordinated sampling can be achieved using random sampling with a cryptographic random number generator, and (3) secure coordinated sampling requires a cryptographic pseudo-random function, keyed with a secret key that should be changed each time the sampler leaks information to the hosts.

I. Introduction

In packet sampling, statistical (or other) techniques are used to sample subsets of packets from the traffic flowing through a network element. The sampled subsets are then used to obtain statistics that characterize the traffic, and are used for an range of purposes, including traffic engineering, troubleshooting, billing and intrusion detection. When sampling is used to determine the traffic mix or for billing purposes, it sufficient to collect data at a single link. However, for certain applications it is necessary to combine data collected in a coordinated manner at multiple vantage points in the network. For example, in Trajectory Sampling [2], where the spatial path of certain packets is traced through a network, statistics are combined from different network elements that sample the *same* subset of packets via coordinated sampling. Furthermore, in passive path-quality measurement, packet loss rates and delay statistics are inferred by sampling packets in a coordinated manner at the ingress and egress ports of a network.

A framework for packet sampling: In Figure 1 we outline a framework for packet sampling, following PSAMP, the IETF Packet Sampling working group [1], [3]. Packet sampling proceeds in three phases: In the *selection process*, the Sampler takes in a data packet stream and selects a subset of the packet stream as an output. Here we model this process with a (possibly randomized) algorithm $\mathsf{Samp}(d)$ that is equal to 1 if a data packet d is sampled and 0 otherwise. In the *reporting process*, the Sampler creates a report stream using relevant statistics (*e.g.* packet contents, packet arrival time, etc.) from the selected packet stream. Finally, during the *export process*, the Sampler sends an export packet, containing the outcome of the reporting process, to the Collector which aggregates statistics from various Samplers and performs analysis on the aggregated data. Export packets are typically sent via the same network that carries the traffic observed by the Sampler. Following the PSAMP working group [1], in this paper we focus on *passive* sampling techniques that do not modify or mark sampled packets. Thus, while the selection process must run at line rate, the Sampler may be implemented as a separate packet monitor that taps off a link, as in Figure 1.

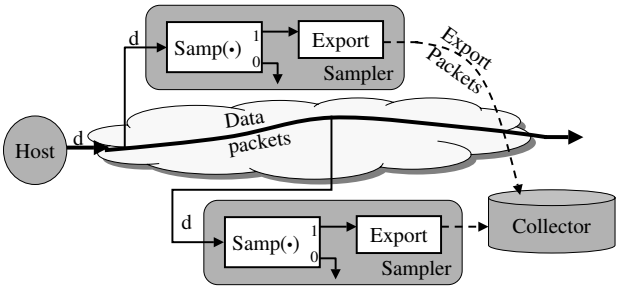

Fig. 1. A framework for packet sampling.

The presence of adversaries: The design of robust packet sampling schemes is complicated by the presence of hosts on the network that have a *strong incentive to behave adversarially in order to cause the* Sampler *to disproportionately select their packets*. Consider these natural scenarios:

- Greedy customers have an incentive to generate packet streams that evade selection by the Sampler in order to avoid being billed by providers for network utilization.
- Malicious users or botnets performing a denial of service (DoS) attack have an incentive to generate packets that evade selection by the Sampler in order to avoid an intrusion detection system.
- Malicious users or botnets may attempt to flood the

network with export packets, or overload and crash the Sampler itself, by crafting packet streams that are selected with 100% probability.

We say that a *packet sampling scheme is secure if no adversarial host can generate a packet stream from which packets are disproportionately selected by the* Sampler, (*i.e.* if the sampling rate is p, then each packet in a packet stream is selected by the Sampler with probability p, regardless of the selection history of all previous packets). We will assume that the host has complete control over all fields of the packets that he sends. We do not consider techniques for preventing a host from spoofing fields in a packet; we are only concerned with ensuring that the Sampler obtains an accurate estimate of the packets actually traversing a network element.

A. Packet sampling techniques

In this paper, we analyze a subset of the packet-sampling techniques considered by the IETF PSAMP working group [4]. We assume that the packet sampling rate is p. We consider two types of uncoordinated sampling, where each Sampler samples packets independently of all other Samplers:

1) *random sampling*, where each packet d is sampled randomly with uniform probability p, independent of packet contents (*i.e.* for all data packets d, $\mathsf{Samp}(d) = 1$ with probability p).
2) *deterministic periodic sampling*, where a packet d is selected based on either arrival time (*e.g.* packets arriving every T ms are selected), or packet count (*e.g.* the T^{th} packet to arrive at the Sampler is selected).

In coordinated sampling, Samplers at different vantage points in the network sample the *same* subset of packets by selecting packets a deterministic manner that is completely dependant on packet contents. We consider the following *passive* coordinated sampling scheme, that does not require the Samplers to modify packets:

3) *hash-based sampling*, where each packet d is sampled if the hash of the packet d falls within a selection range

$$\mathsf{Samp}(d) = \begin{cases} 1, & f(d) \in [R_1, R_2]; \\ 0, & \text{otherwise.} \end{cases} \quad (1)$$

where f is a hash function taking on values in $\{1, ..., 2^n\}$ and the size of the selection range $[R_1, R_2]$ is a p-fraction of the size of the output range of the hash function (usually $p\,2^n$). Hash-based sampling may also be performed using a keyed hash function f_k, where k is a secret key known only to the Samplers. That is, a packet d is selected if $f_k(d) \in [R_1, R_2]$.

B. Overview of our results and recommendations

We find that the following packet sampling schemes are vulnerable[1] to attacks by adversarial hosts:

- *Deterministic periodic sampling.* (Section II-B.)
- *Unkeyed hash-based sampling,* even with a cryptographic hash function and secret selection range $[R_1, R_2]$ unknown to the host. (Section III-A.)
- *Keyed hash-based sampling using a non-cryptographic hash function*, even with a secret hash key k and secret selection range $[R_1, R_2]$. (Section III-B.)

While the current drafts of the PSAMP charter suggest that last two (hash-based) schemes may be used for secure packet sampling, our results indicate that these schemes have serious security vulnerabilities.

We make the following recommendations for secure packet sampling in the presence of adversarial hosts:

- *Random sampling* should be used for uncoordinated sampling using a cryptographically-strong random number generator. (Section II-A) In practice, a cryptographically-strong random number generater can be can be realized using a stream cipher, (*e.g.* RC4), or using a block cipher in counter mode (*e.g.* AES in counter mode).
- *Keyed pseudorandom function (PRF)-based sampling* with a secret key k and a publicly-known selection range $[R_1, R_2]$ should be used for passive coordinated sampling. (Section IV-B.) In practice, pseudorandom functions can be implemented using cryptographic hash functions. For example, fixed- or variable-length-input PRF's can be realized using a block cipher in cipher-block-chaining mode (*e.g.* AES in CBC mode) [5]. A PRF that takes in fixed-length inputs can also be realized using an cascaded cryptographic hash function (like MD5 or SHA1), where the secret key is used as the intial vector [6]. To realize a PRF that takes in variable-length inputs, the "append-cascade" construction of [6] should be used; the secret key should be split into two parts $k = (k_1, k_2)$, and a cascaded cryptographic hash function (like MD5 or SHA1) should take in k_1 as an intial vector, and post-pend k_2 to the input.
  Furthermore, to protect the system from adversarial hosts that attempt to break security by repeatedly sending identical packets in a *replay attack*, we further recommend (see Section IV-C) the following:
  * Export packets sent from Sampler to Collector should be secured (see Section I-C for details).
  * The secret key to the PRF should be changed every so often to limit the time window of vulnerability to replay attacks. For example, the PRF key could be changed each time any side channel (*e.g.* a bill sent to a customer) leaks information to the host about the past history of packets selected by the Sampler.

We believe that these modest recommendations could readily be implemented in practical packet sampling schemes. The high-throughput PRFs required for coordinated sampling can be efficiently implemented in pipeline hardware. Furthermore, since the random number generator required for uncoordinated sampling is typically implemented using cryptographic techniques of comparable complexity to a PRF, PRFs-based coordinated sampling may be sufficient for most applications.

[1] These vulnerabilities exist even when there is no leakage of information to the adversarial host about the past history of packets selected by the Sampler.

TABLE I

SYMBOLS USED IN THIS PAPER.

p	Sampling rate
d	Data packet
$f(\cdot)$	Hash function produces outputs in the set $\{1, ..., 2^n\}$
k	Hash key (in hash-based sampling)
$[R_1, R_2]$	Selection range (in hash-based sampling)
T	Sampling period (in deterministic sampling)
H	Past history of selected packets

Finally, updating the PRF key with a new session key derived from a master secret shared by the Samplers can be done without incurring much additional management overhead (*e.g.* using a 'forward secrecy' protocol as in the Internet Key Exchange (IKE) [7]),

C. Past history: Export packets and other side channels

An adversarial host can often use information about the past history of packets selected by the Sampler to learn how to break the security of a packet sampling scheme. Past history information may be obtained by eavesdropping on the export packets sent over the network from a Sampler to a Collector. Notice that even if the export packets are encrypted, and adversary might still be able to use timing information to determine whether or not a packet is sampled (*e.g.* if export packets are released immediately after a packet is sampled, the adversary can use timing to learn which packets where sampled). Furthermore, the length of the export packet can also leak information about the packets or number of packets sampled by the device. Other 'side channels' can also leak information about selected packets. For example, an adversarial host can monitor his billing information to learn the fraction of the packets that he sent that were also selected by the Sampler.

Securing the export packets: A simple approach to prevent export packets from leaking information is to do away with them altogether; that is, to physically co-locate the Sampler and Collector, so that export packets need not be sent via the network. However, this solution in infeasible in coordinated sampling applications that require the Collector to aggregate statistics from Samplers at different physical locations in the network. Alternatively, to prevent export packets sent via the network from leaking information, they should be encrypted, padded to constant length, and sent out a fixed time intervals.

The effect of past history on security: When an unkeyed hash function or a non-cryptographic keyed hash function is used for sampling, an adversarial host can use as small amount of past history information to not only break the security, but also to *learn the secret* selection range and hash key used by the Sampler (Sections III-A and III-B). Furthermore, in Section IV-C we discuss how past history information can be used by adversarial host to launch replay attacks in PRF-based sampling.

II. UNCOORDINATED SAMPLING

A. Random Sampling: Security

In random sampling, each Sampler flips an independent, p-biased coin to decide whether or not it selects a packet. Because each Sampler *independently* decides whether or not to select a packet, random sampling can only be used for uncoordinated packet sampling. Random sampling is secure even when an adversarial host knows the entire past history of packets selected by the Sampler. This follows from the fact that each packet is selected independently with probability p, independent of packet contents, and thus independent of past history. Random sampling requires a cryptographically-strong random number generator for packet selection (in practice, a stream cipher, or a block cipher in counter mode may be used); otherwise, it is possible for an adversary to predict the result of the $\mathsf{Samp}(\cdot)$ function with better probability than just guessing randomly with probability p. Notice that security fails if the $\mathsf{Samp}(\cdot)$ function uses a number generator the produces a pattern of numbers that the adversary can predict (*e.g.* a deterministic sequence that repeats the same pattern of numbers, *e.g.* the Quasi-Random Signal Sequence (QRSS)).

B. Deterministic Periodic Sampling: Vulnerabilities

In deterministic periodic sampling, a packet is selected (in count-based sampling) if it is the T^{th} packet to arrive at the Sampler, or (in time-based sampling) if it arrives at the Sampler during the sampling time interval (of length pT) that repeats every T seconds. We claim that an adversarial Host who knows the sampling period T can easily break security of deterministic periodic sampling, even when there is no leakage of past history information. Here we only describe an attack on time-based sampling. Assume that Host knows the sampling period T. Then, Host simply chooses a random time instant t_o to send his first packet, and then proceeds to send his next packet at time $t_o + T$. He continues in this manner so that his packets are sent spaced out at time intervals of T. With high probability $1 - p$, the random start time t_o will *not* coincide with the sampling interval used by the Sampler. As such, with high probability, the Sampler will not select the any of the packets sent by the Host, and the scheme is not secure.

III. VULNERABLE COORDINATED SAMPLING SCHEMES

Recall that in hash-based coordinated sampling, a packet d is selected if its hash $f(d)$ falls in a selection range $[R_1, R_2]$, (see Equation 1). In this section, we start by presenting attacks by adversarial hosts that that break the security of unkeyed-hash-based sampling, even when the hash function is cryptographic and the selection range is kept secret. We then show attacks that break the security of non-cryptographic keyed-hash-based sampling, even when the hash key k and selection range $[R_1, R_2]$ are kept secret. These attacks suggest that a secure coordinated packet sampling scheme should use a keyed hash function, and furthermore, that the keyed hash function should be cryptographically strong. (In Section IV-B

we shall present a security argument that shows that keyed-hash-based sampling using cryptographically-strong pseudo-random functions is indeed secure.)

Cryptographic hash functions: In this paper, we idealize an unkeyed cryptographic hash function f as a publicly known *truly random function*[2] which can be thought of as the following: for each input x, a value in $\{1, \ldots, 2^n\}$ is chosen uniformly at random to be $f(x)$. In practice, we can think of MD5, SHA1 or the hash functions in [9] as cryptographic hash functions. For our purposes, a (keyed, or unkeyed) hash function that is 'not cryptographically strong' is a hash function that *can* be efficiently distinguished from a random function. It follows that any hash function that is easy to invert (*i.e.* if given some output y it is easy to find an input x such that $f(x) = y$) is also not cryptographically strong. Our canonical example of a non-cryptographic keyed hash function will be the CRC keyed with a secret modulus k, which we model as

$$f_k(d) = d \mod k \qquad \text{(CRC)}$$

Observe that because the CRC is a linear function, it can easily be distinguished from a random function by checking if $f_k(d+1) = f_k(d) + 1$ for arbitrary d. We will also consider the non-cryptographic keyed hash function

$$f_k(d) = ad + b \mod k \qquad (2)$$

where the secret key is the parameters a, b, k. This hash function can also be thought of the CRC of packet d post-pended with a secret string (*e.g.* CRC$(packet, secretString)$). (The BOB hash function described in the PSAMP charter [1] uses some variant of this approach.)

A. Unkeyed Hash Based Sampling: Vulnerabilities

We now show attacks on unkeyed-hash-based packet sampling with a cryptographically-strong hash-function and secret selection range. In our first attack, the adversarial host breaks security without using any information about the past history of selected packets. While the first attack is serious enough to break the security of the packet sampling scheme, we also present a second (more serious, but perhaps less realistic[3]) attack in which the Host uses a very small amount of past history information (*e.g.* billing information) to not only break security, but also to learn the secret selection range.

Attack without past history: To attack, Host starts by choosing a random number $\hat{R}_1 \in \{1, 2, ..., 2^n\}$, and setting $\hat{R}_2 = \hat{R}_1 + p\,2^n$. Then, Host does the following for each packet that he wants to send to Sampler: Host chooses d. Then Host computes $f(d)$. If he finds that $f(d) \in [\hat{R}_1, \hat{R}_2]$, he sends d to Sampler (since we model f as a truly random function, this happens with probability p). Otherwise, he tries the above again with a newly chosen packet d'. Using this process, Host ensures that the hash of each packet he sends falls in the range $[\hat{R}_1, \hat{R}_2]$. Now observe that with high probability $1-2p$, $[\hat{R}_1, \hat{R}_2]$ will be different from the true private selection range $[R_1, R_2]$ used by Sampler, so that none of Host's packets will be selected by Sampler, and Host breaks security. Notice that for each packet that Host sends to Sampler, then with 99% probability Host needs to try at most $\frac{\log(0.01)}{\log 1-p}$ packets before he finds one that falls in his chosen range $[\hat{R}_1, \hat{R}_2]$ (*e.g.* if $p = .02$, then with 99% probability he needs to compute at most 228 hashes). Furthermore, if the hash function is not cryptographically strong, the adversary needs to compute even fewer hashes for each packet he sends (*e.g.* if the hash function is the CRC function with a publicly-known key k, then the adversary can immediately choose packets d such that $d \mod k \in [\hat{R}_1, \hat{R}_2]$).

Learning the selection range using past history: We will assume that during every billing interval Host obtains a bill that indicates whether or not he sent packets over the network during that interval. (This past history information could also have been obtained for unsecured export packets or other side channels.) To learn Sampler's secret selection range $[R_1, R_2]$, Host does the following: During each billing interval, Host chooses a new range $[\hat{R}_1, \hat{R}_2]$ and crafts packets such that the hash of each packets falls in $[\hat{R}_1, \hat{R}_2]$ as described in the attack above. If Host is not charged during the billing interval, then he learns that $[\hat{R}_1, \hat{R}_2]$ is disjoint from the true selection range $[R_1, R_2]$. Similarly, if he is charged, it follows that $[\hat{R}_1, \hat{R}_2]$ overlaps with $[R_1, R_2]$. This process can be repeated (with cleverly selected ranges $[\hat{R}_1, \hat{R}_2]$) during each billing interval until Host eventually learns the secret selection range $[R_1, R_2]$.

B. Non-Cryptographic Keyed Hash Based Sampling: Vulnerabilities

Because of the large variety of non-cryptographic hash functions, in this section we focus on the CRC, and the hash function $f_{a,b,k}(d) = ad + b \bmod k$ as examples of non-cryptographic keyed hash functions. We assume that the key k (and parameters a, b) and the selection range $[R_1, R_2]$ are kept secret. As in Section III-A, we show an attack on the CRC where the adversarial host breaks security without using any past history information, and another attack where the Host uses past history to learn the secret key and selection range. We then show how an adversary can use a small amount of past history information to break the security of scheme based on the hash function $f_{a,b,k}(d) = ad + b \bmod k$.

We will think of a data packet d as an integer (using the obvious mapping from bits to integers). We will assume a lower bound on the size of the hash key of $k > 2^{n-m}$ for the purposes of our analysis, and to ensure that the output range of the hash function is sufficiently large.

1) Attack on the CRC

Attack on CRC without past history: This attack is very similar to the attack without past history in Section III-A, except that here, the Host does not know know the hash key

[2]More precisely, we model the hash as a random oracle, see [8].

[3]These attacks may be more realistic than they first appear, particularly if the Host knows that only the first N bytes of each packet are used as input to the hash function. Then, the Host can break security by crafting packets as shown in these attacks, while still getting useful communication from the remaining bytes of each packet.

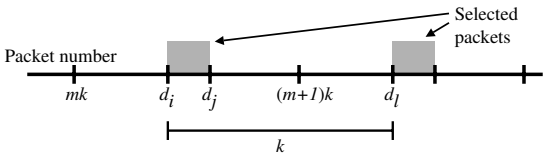

Fig. 2. Attack on the CRC with secret modulus.

k, and instead relies on the linearity of the CRC function. To attack, Host starts by choosing a packet d, and then sends a linear progression of packets d, $d+1$, ..., $d + p\,2^{n-m}$ to Sampler. Observe that by the linearity of the CRC, all the packets sent by Host will fall in the range $[\hat{R}_1, \hat{R}_2]$ where $\hat{R}_1 = d \mod k$ and $\hat{R}_2 = \hat{R}_1 + p\,2^{n-m}$. As in Section III-A, with probability $1 - 2p$ the range $[\hat{R}_1, \hat{R}_2]$ will be different from the true private selection range $[R_1, R_2]$, so that none of Host's packets will be selected and Host breaks security.

Learning the selection range and the CRC hash key using past history: As in Section III-A, we will again assume that at the end of every 'interval' Sampler leaks information (because of billing, or export packets) about whether or not a packet was selected. We sketch the method Host uses to learn Sampler's secret information as follows: Host starts by choosing a data packet d_1. In the first interval Host sends packet d_1, and checks at the end of the interval if d_1 was selected. During the next interval, he sends $d_2 = d_1 + 1$ and checks if d_2 was selected. He continues this process until he finds packets d_i, d_j, d_ℓ such that

- d_i is not selected and d_{i+1} is selected
- d_j is selected and d_{j+1} is not selected, and $j > i$
- d_ℓ is not selected and $d_{\ell+1}$ is selected and $\ell > j > i$.

Now, observe from Figure 2 that because the CRC is linear, the transition between selected packets and unselected packets occur at the edges of the true selection range $[R_1, R_2]$, and that selection ranges are separated by an additive distance of k. It follows the adversary can learn the key and selection range as follows:

$$\begin{array}{rcl} k & = & d_\ell - d_i \\ R_1 & = & d_i \mod k \\ R_2 & = & d_j \mod k \;=\; R_1 + pk \end{array} \tag{3}$$

Note that in practice this attack should be performed using more efficient search techniques to find, d_i, d_j, d_ℓ (instead of simply incrementing d by 1 at each interval). We now present one such an efficient algorithm, based on binary search.

Consider a range of packets $[d, d + 2^{n+1}]$, and divide the range into sub-ranges of width $p2^{n-m}$. It follows that at least two of these (non-adjacent) sub-ranges will overlap with the two selection ranges shown in Figure 2. Thus, the Host can perform two binary searches[4] through the range $[d, d + 2^n]$ until he finds two non-adjacent sub-ranges where his packets are selected. Notice that once the Host finds these two sub-ranges, the host can obtain an estimate (within $\pm p\,2^{n-m}$) of the value of the hash key by computing the distance between the two sub-ranges. Recalling that the worst case running time of a binary search through N items is $1 + \log_2 N$, it follows that each sub-range can be found within

$$1 + (m+1) + \log_2 \tfrac{1}{p} \tag{4}$$

time intervals. To more accurately determine the secret key, the Host can use two more binary searches to find the edges of the two selection ranges (*i.e.* d_i, d_ℓ in Figure 2). From the previous search, Host knows two packets d_{sel} and d'_{sel} that overlap with the true selection range $[R_1, R_2]$. Host can now use a binary search around d_{sel} (resp. d'_{sel}) to find the edge of the first selection range d_i (resp. d_ℓ).[5] Since the host Host must search through at most $pk < p2^n$ possible packets to find d_i, each binary search can be completed after

$$1 + n + \log_2 p \tag{5}$$

time intervals. Thus, once Host learns d_i, d_ℓ, he can exactly learn the secret key k and selection range $[R_1, R_2]$ using Equation 3 within

$$6 + 2(n+m) \tag{6}$$

time intervals.

If we assume that the output of the hash function is $n = 32$ bits (*i.e.* CRC32), the sampling rate is $p = 1\%$, and that $m = 8$ bits, it follows that it takes Host 34 time intervals to approximately learn the key (per Equation 4) and 86 time intervals to exactly learn the key (per Equation 6). In more concrete terms, the time it takes for the Host to learn the secret key depends strongly on how the Samp leaks past history information to the Host. That is, if the Host is using monthly billing information to learn past sampling outcomes, then a 'time interval' is one month, and the Host needs a few years to learn the key. On the other hand, if Host is

[4]This binary search proceeds as follows. During each time interval, the host starts with a range of packets of width W (starting with the range $[d, d+2^{n+1}]$ of width $W = 2^{n+1}$) and sends packets that fall only in the *left half* of the range(*e.g.* in $[d, d + 2^n]$). More specifically, during each time interval Host sends at least $2\frac{W}{p2^{n-m}}$ packets that are equally spaced in the left half of the range of width W. Then, if at the end of the time period Host learns that that some of his packets are sampled, then the true selection range is somewhere in left half of the range he is searching through, and during the next time interval Host repeats the search process in the left half of the range. Otherwise, during the next time interval Host repeats this process in the right half of the range. The process continues until the host has found a range of width $W = p\,2^{n-\ell}$ where some of Host's packets are selected.

[5]This binary search proceeds as follows. Host knows that some packet d_{sel} falls within the first selection range. During each time interval, Host starts with some range of packets (starting with the range $[d_{\mathsf{sel}} - p2^n, d_{\mathsf{sel}}]$ of width $W = p\,2^n$) and sends a packets that fall only in the *left half* of the range (*e.g.* $[d_{\mathsf{sel}} - p2^n, d_{\mathsf{sel}} - p2^{n-1}]$). Then, if at the end of the time interval Host learns that that some of his packets were selected, then d_i (the edge of the selection range) is somewhere in left half of the range of packets the Host is searching through, and in the next time interval Host repeats the search process in the left half of the range. Otherwise, in the next time interval, Host repeats the search process in the right half of the interval. The search continues until Host finds the edge of the selection range d_i.

obtaining information from export packets that are transmitted each second, then the Host can learn the key in a few minutes.

2) Attack on the hash function $\mathbf{f(x) = ax + b \mod k}$

Breaking security using $O(\frac{1}{p})$ past history: We now show how an adversarial Host can break the security of a packet sampling scheme that uses the sampling function

$$\mathsf{Samp}(d) = \begin{cases} 1, & \text{if} \quad ad + b' \mod k \in [R_1, R_2]; \\ 0, & \text{else.} \end{cases}$$

Where a, b', k, R_1, R_2 are all secret (and therefore unknown to the adversarial Host). Notice that without loss of a generality, we can re-write the sampling function as

$$\mathsf{Samp}(d) = \begin{cases} 1, & \text{if} \quad ad + b \mod k < pk; \\ 0, & \text{else.} \end{cases} \tag{7}$$

where a, b, k are all unknown to the adversarial Host, and p is the known sampling frequency. We will sometimes refer to hash function $f(d) = ad + b \bmod k$. In the rest of this section we shall show how *an attacker that observes $O(\frac{3}{p})$ past sampling outcomes to find two packets that* were *selected by the sampler, can craft a stream of $O(\frac{1}{p^2})$ packets that* will not be *selected by the sampler.* [6]

The attack: The attack proceeds as follows. First, the adversary uses past history information to finds two packets, d_1 and d_2, such that

- d_1, d_2 are both even numbers.
- d_1 and d_2 are both selected by the sampler
- $\Delta = \frac{|d_2 - d_1|}{2}$ is not selected by the sampler

We argue below that finding such d_1, d_2 requires knowledge of, on average, $\frac{3}{p}$ past sampling outcomes. Next, the adversary sets $\delta = |d_2 - d_1|$, and sends a packet stream of the form

$$(2i+1)\Delta,\ (2i+1)\Delta + \delta,\ ...,\ (2i+1)\Delta + \ell_i\delta \tag{8}$$

for $i = 0, ..., \lfloor \frac{1}{2}(\frac{1}{p} - 1) \rfloor$, and $\ell_i = \lfloor \frac{1}{2}(\frac{1}{p} - 2i - 1) \rfloor$. We show below that the this approach allows the adversary to craft a stream of $\approx \frac{1}{8p^2}$ packets that will not be selected by the sampler. Thus, if $p = 1\%$, the adversary can craft a packet stream of 1325 packets if can find the required d_1, d_2 packets.

Why the attack works: Observe from Fig. 3 that since d_1, d_2 are both fall within the small selection range, then $a(d_2 - d_1) \approx nk$ where n is an integer. If Δ was *not* be selected by sampler, it follows that with high probability n is odd so that

$$f(\Delta) = a\Delta + b \mod k = \tfrac{k+\epsilon}{2} + b \tag{9}$$

where $\epsilon \in [-pk, pk]$. (Observe that if Δ was selected by sampler, the halfway point between d_1 and d_2, Δ, falls within some selection range so that with high probability n is even.) Therefore, in expectation, the adversary needs needs to observe

[6]Knowledge of more past history of sampling outcomes can allow the adversary to craft even longer packet streams. Furthermore, more knowledge of sampling outcomes can be used to completely learn the secret parameters a, b, k. While we do not describe these attacks here, they use techniques similar to those used by [10], [11] to solve the "hidden number problem".

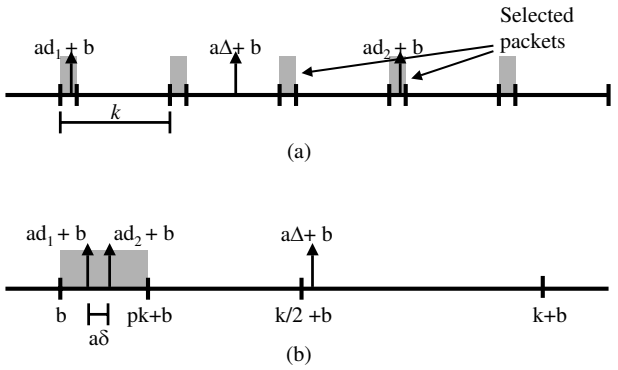

Fig. 3. Attack on keyed hash-based sampling with $f_{a,b,k} = ad + b \bmod k$. Position of d_1, d_2, Δ and δ (a) along the integers (b) along the integers $\bmod k$.

sampling outcomes for, on average, $\frac{3}{p}$ randomly selected packets in order to find d_1, d_2 as described above. [7]

Now, since d_1, d_2 are both selected by the Sampler, it follows that that $-pk \bmod k < a(d_1 - d_2) \bmod k < pk$. If we define $\delta = d_2 - d_1$ it follows that

$$f(d + \delta) = f(d) + \epsilon' \tag{10}$$

where $\epsilon' \in [-pk, pk]$. [8] We now show why none of the packets crafted by the adversary in Equation (8) will be selected by the sampler. First, we observe that for each packet of the form $(2i+1)\Delta$ in the crafted stream, it follows that

$$f((2i+1)\Delta) = \tfrac{k + (2i+1)\epsilon}{2} + b$$

where $\epsilon \in [-pk, pk]$ and the equality follows from Equation (9). As long as $\frac{1}{2}(2i+1)\epsilon < \frac{k}{2}$, no packet of the form $(2i+1)\Delta$ will be selected, which is the case when $i = 0, ..., m$ and

$$m = \lfloor \tfrac{1}{2}(\tfrac{1}{p} - 1) \rfloor \tag{11}$$

Now, fix i, and observe that for each packet $(2i+1)\Delta + j\delta$ in the crafted stream, it follows that

$$f((2i+1)\Delta + j\delta) = \tfrac{k + (2i+1)\epsilon}{2} + b + j\epsilon'$$

where $\epsilon, \epsilon' \in [-pk, pk]$ and the equality follows from Equations (9) and (10). Now, to ensure that no packets of the form $\Delta + j\delta$ will be selected, we must have that $\frac{1}{2}(2i+1)\epsilon + j\epsilon' < \frac{k}{2}$ which is the case as long as for each i, $j = 0, 1, ..., \ell_i$ where

$$\ell_i = \lfloor \tfrac{1}{2}(\tfrac{1}{p} - 2i - 1) \rfloor \tag{12}$$

It follows that the length of the packet stream crafted by the adversary is

$$\sum_{i=1}^{m} \ell_i + 1 = \sum_{i=1}^{\lfloor \frac{1}{2}(\frac{1}{p} - 1) \rfloor} \lfloor \tfrac{1}{2}(\tfrac{1}{p} - 2i - 1) \rfloor + 1 \quad \approx \quad \frac{1}{8p^2}$$

[7]To find each selected packet, the adversary needs to observe an average of $\frac{1}{p}$ sampling outcomes, and given a selected packet d_1, the adversary needs to observe on average two more selected packets until he finds one that is an even number of multiples of k away from d_1.

[8]We use the observation of Equation (10) in a manner that is analogous to the observation that we used to break the security of CRC-based sampling, *i.e.* for the CRC $f(x + 1) = f(x) + 1$.

3) Attacks on other non-cryptographic keyed hash functions:

Instead of sending a linear progression of packets as in the attacks on the CRC, attacks on other non-cryptographic keyed hash functions require the Host to send a more complicated packet stream that depends on the structure of the hash function. For example, if the hash function is easy to invert (*i.e.* given output y it is easy to find an input x such that $f_k(x) = y$), then the following attack breaks security (by creating a packet stream that is disproportionably sampled): if the Host sends a progression of packets such that $f_k(d_{i+1}) = f_k(d_i)+1$ (*i.e.* Host finds d_{i+1} by inverting the hash function), then with probability $1-2p$ none of Host's packets will be selected, as in the first attack we showed on the CRC.

IV. Secure PRF-based Coordinated Sampling

In this section, we use techniques from theoretical cryptography to argue that coordinated sampling based on keyed pseudorandom functions is secure. To do this, we first formalize the intuitive notion of security that we have been using throughout this paper with a game-based security definition. In cryptography, game-based definitions (consisting of a security game and a security condition) are used to obtain precise guarantees of security (*e.g.* see [12]). Next, we use the game-based definition to argue that PRF-based sampling is secure if the hash key k is kept secret, up to the possibility of replay attacks. Finally, we give practical recommendations for mitigating the effect of replay attacks on PRF-based sampling.

Pseudorandom Functions (PRFs): A PRF is a *deterministic*[9] function that takes in a random secret key k and an input x and outputs a value in $\{1,\ldots,2^n\}$ as $f_k(x)$. A PRF [12] is defined as a keyed function that cannot be distinguished by any computationally-efficient algorithm (that does not know the key) from a truly random function with better than non-negligible probability.

In practice, pseudorandom functions can be efficiently implemented in pipelined hardware using keyed cryptographic functions. For example, fixed- or variable-length-input PRF's can be realized using a block cipher in cipher-block-chaining mode (*e.g.* AES in CBC mode) [5]. A PRF that takes in fixed-length inputs can also be realized using an cascaded cryptographic hash function (like MD5 or SHA1 or the hash functions of [9]), where the secret key is used as the intial vector [6]. To realize a PRF that takes in variable-length inputs, the "append-cascade" construction of [6] should be used; the secret key should be split into two parts $k = (k_1, k_2)$, and a cascaded cryptographic hash function (like MD5 or SHA1 or the hash functions of [9]) should take in k_1 as an intial vector, and post-pend k_2 to the input.

A. Adversarial Hosts: Formal Security Definition

The game setting: The game setting for packet sampling with adversarial hosts, shown in Figure 4, defines the power of the (computationally-efficient) adversary Host and models how he interacts with the honest parties (Sampler, Collector). Host generates data packets d, *subject to the requirement that each packet is unique* (this requirement precludes replay attacks, see our discussion below), and sends data packet d to Sampler who then runs $\mathsf{Samp}(d)$ on each packet. If $\mathsf{Samp}(d) = 1$ then the data packet is sampled and included in the export packet that Sampler sends to Collector. We also give Host the power to eavesdrop on the export packets that Sampler sends to Collector. The game proceeds in two phases: During the training phase, Host is allowed to generate and send data packets, and observe export packets. During the training phase, we keep track of H, the history of packets sampled by Sampler, and export packets sent to Collector. (H contains a list of pairs, where each pair $(d_i, \mathsf{Samp}(d_i))$ records the i^{th} data packet sent by Host d_i, and whether or not Sampler selected d_i.) Then, during the challenge phase, Host must generate a challenge data packet d^*.

Security condition: We say that Host *wins the packet sampling game* (*i.e.* breaks the security the packet sampling scheme) if, conditioned on the past history H, the challenge packet d^* is sampled with probability different from the sampling rate p. That is, for some negligibly small value of ϵ, we say the packet sampling scheme is secure if, for all possible Host playing the packet sampling game, then

$$\mid \Pr[\mathsf{Samp}(d^*) = 1 | H] - p \mid \leq \epsilon$$

Replay Attacks: In a hash-based sampling protocol, when Host learn (*e.g.* from export packets or other side channels) whether or not some packet d was selected by the Sampler, then Host can trivially break security by re-sending d.[10] We call this a *replay attack*. Replay attacks may be a significant threat when only a portion of the fields in a packet is used to decide if a packet should be selected; a user may replay those portions of the packet while using the rest of the packet to carry his (non-replayed) communication payload without being detected by the Sampler. Malicious hosts launching a DoS attack on a network may also attempt to flood a network with replays of a packet that they learned was not selected by Sampler. Because our game-based definition restricted the Host to sending only unique packets, our security game does *not* consider replay attacks (*i.e.* a scheme that satisfies our security definition may still be vulnerable to replay attacks). We take this approach because, in a deterministic packet sampling scheme, completely preventing replay attacks requires the Sampler to append a unique identifier to each packet as it enters the network. However, since we are interested in *passive* sampling protocols that do not modify packets, tagging packets in not a viable solution.[11] Thus, while our formal security definition does not preclude replay attacks, we discuss practical techniques for mitigating the effect of replay attacks in Section IV-C.

[9] All the randomness in the PRF comes from the choice of secret key.

[10] More formally, if past history H contains the pair $(d, 1)$, then if Host knows to send $d^* = d$ as his challenge packet then d^* is selected with probability 1 and Host breaks security.

[11] Another way to completely prevent replay attacks in a deterministic scheme is to change the sampling scheme's parameters, *e.g.* hash key, each time a packet is sampled. This approach is, again, too impractical to consider here.

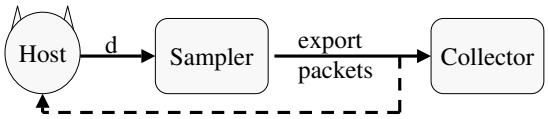

Fig. 4. Packet sampling with adversarial hosts.

B. Security of PRF-based Sampling

We claim that keyed cryptographic pseudorandom function (PRF) based sampling satisfies the game-based security definition of Section IV-A. The key k to the PRF f_k must be kept secret. The selection range $[R_1, R_2]$ need not be kept secret. To prove this, we consider two different packet sampling games; Game G_{prf} is the packet sampling game (Section IV-A) with PRF hash-based sampling and publicly known sampling range $[R_1, R_2]$, and Game G_{rand} is the same packet sampling game but with random sampling. Consider an adversary Host playing game G_{rand}, where for every packet d that Host gives to Sampler during game G_{rand}, Sampler decides to sample d with truly random probability p. Note that game G_{rand} is identical to game where Sampler generates a new, truly random number for each packet d that Host gives him, and decides to sample d if the random number falls in the publicly known selection range $[R_1, R_2]$. Next, consider Host playing game G_{prf}, where for each packet d, Sampler generates a new pseudorandom number $f_k(d)$, and checks if the pseudorandom number falls in the selection range $[R_1, R_2]$. Recall that our security game in Section IV-A restricts the Host to sending only distinct data packets d to Sampler. It follows that in game G_{prf} Sampler will generate a new, distinct pseudorandom number for every data packet that Sampler receives. Now by the definition of PRFs, the random numbers generated by the PRF will be indistinguishable (by any computationally-efficient algorithm) from truly random numbers. Therefore, since each for each packet in Game G_{prf} Sampler generates a new pseudorandom number that is indistinguishable from a truly random number, it follows that game G_{prf} is (computationally) indistinguishable from game G_{rand}. (Notice that if games G_{prf} and G_{rand} could be distinguished by some computationally-efficient algorithm, we could use that algorithm to construct an algorithm that distinguishes between PRF and truly random functions, and therefore break the security of the PRF.) Now because game G_{rand} is secure even when the adversary has access to the entire past history H (see Section II-A), it follows that game G_{prf} is also secure, so that PRF-based sampling is secure.

C. Mitigating Replay Attacks in PRF-based Sampling

Replay attacks are particularly problematic when the host has access to past history of selected packets (because he can use this history to figure out exactly which packets he should replay to, say, avoid being detected by the Sampler). Thus, one way to mitigate the impact of replay attacks over a long time scale is to *change the PRF key k each time past history information leaks out of the* Sampler. A forward secrecy protocol, *e.g.* as in the IKE [7], can be used to update the PRF key with a session key derived from a master secret shared by the Samplers. When the PRF key is changed, past history information becomes useless; that is, since from the properties of PRFs, any packet d will selected with a new PRF key independently of whether or not d was selected by the old PRF key. In most systems, past history information leaks out of the Sampler on a relatively slow timescale (*e.g.* every time an unsecured export packet is sent, or each time a bill is sent). Thus, a practical approach to mitigating the effect of replay attacks is to prevent export packets from leaking information (by securing them as described in Section I-C), and to change the PRF-key each time other side channels (*e.g.* billing) leak information to the hosts.

However, short-time-scale replay attacks are still possible! We emphasize here that securing the export packets and changing the PRF key does *not* prevent an adversarial Host from *blindly* performing the following replay attack: Host sends N consecutive identical packets to the Sampler in hopes that all N of his packets will not be selected. Recall that (uncoordinated) random sampling is not vulnerable to these blind replay attacks. Again, the PRF key can be changed to limit the time window of vulnerability to these blind replay attacks.

V. Other security issues

In the previous sections, we considered the security of the packet sampling system in Figure 1 against adversarial hosts that attempt to craft packets that are disproportionately selected by the Sampler. We now consider the security of the system against adversaries that try to 'game' the system by tampering with the export packets, and adversarial routers inside the network that attempt to bias passive measurement.

A. Authenticity of Export Packets

In Section I-C we argued that to prevent leakage of information about the past history of selected packets to the adversary, then export packets must be encrypted, padded to fixed length, and send out at constant time intervals. We now call attention to the fact that some entities on the network may have an incentive to forge the export packets sent from Sampler to Collector (see Figure 1). For example, during a DoS attack, an adversary may forge export packets to trick the Collector into thinking that the network is operating under normal conditions. Fortunately, there are simple solutions to this problem. One approach is to secure the infrastructure itself, by configuring access control lists at the perimeter of the network to filter all the packets destined to a Collector that originate outside the network. When the infrastructure cannot be secured (*e.g.* because adversarial entities may be located *inside* the network) then export packets should be authenticated (using a symmetric- or public-key signature [12]) to prevent an adversary from modifying information sent between Sampler and Collector.

B. Secure Passive Path Measurement

As discussed in [3], another application of coordinated packet sampling is to allow network entities to measure packet

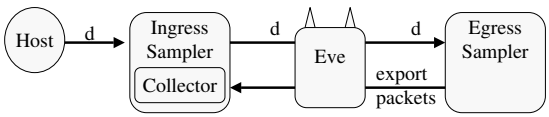

Fig. 5. Passive path measurement in the presence of adversaries (Eve).

loss and delay through the network. In passive path measurement, different Samplers sample the same set of packets; information is aggregated at the Collector, and then packet loss and delay are calculated (*e.g.* by checking if a packet seen by an ingress Sampler was dropped before it reached an egress Sampler). Passive measurement can be used for troubleshooting purposes, or to inform routing decisions. A passive path measurement scheme (especially one designed for the interdomain setting) must be robust to *adversarial routers along the data path* that attempt to bias path loss measurements by dropping packets in a manner that is not detected by the measurement system.

We studied this problem, which we called *fault detection*, in [13]. The setting is reproduced in Figure 5. Instead of considering an adversarial Host, here we consider an adversarial router (aka Eve) on the data path between the ingress Sampler (aka Alice) and the egress Sampler (aka Bob). The adversarial router Eve wants to drop packets without being detected. Eve also eavesdrops on the export packets sent between the Sampler and Collector. In [13], we proposed two new fault detection protocols, *Pepper Probing* and *Salt Probing*, for secure fault detection that are compatible with the packet sampling framework of Figure 1. Both Pepper Probing and Salt Probing use a pseudorandom function to sample packets in a coordinated manner at the ingress and egress Sampler. Authenticated export packets, containing lists of the packets selected by egress Sampler, are sent to a Collector co-located with the ingress Sampler, as shown in Figure 5. In [13] we show that both Pepper Probing and Salt Probing are secure even if the adversarial router has access to unsecured export packets. The results of [13] again confirm the importance of pseudorandom functions in secure coordinated sampling.

VI. Conclusions

Vulnerable Sampling Techniques: We showed that, even when an adversarial host has no information about the past history of packets selected by the Sampler, the host can break security by crafting packets that are disproportionably selected by the Sampler in (1) deterministic periodic sampling, (2) unkeyed hash-based sampling, and (3) non-cryptographic keyed hash based sampling. We also showed how an adversary can use information obtained from export packets or billing to learn the secret selection range (and hash key) in unkeyed hash-based sampling, and non-cryptographic keyed hash-based sampling.

Recommendations for Secure Sampling: For secure uncoordinated sampling, we recommend the use of random sampling with a cryptographically-strong random number generator (*e.g.* a stream cipher, or a block cipher like AES in counter mode). Random sampling schemes are not subject to replay attacks, even when export packets are not secured. For secure coordinated sampling, we recommend the use of a cryptographically-strong pseudorandom function (PRF) keyed with a secret key and with a publicly known selection range. We emphasize that a PRF operating at router line rates can be efficiently implemented using pipelining in hardware (*e.g.* using a block cipher in CBC mode, or using MD5, SHA1, or the hash functions from [9] keyed in append-cascade mode [6]). We further emphasize that the hash function $f_k(d) = d \bmod k$ (the CRC) and $f_{a,d,k}(d) = ad + b \bmod k$ should *not* be used to implement a PRF.

Furthermore, despite the fact that passive PRF-based sampling remains vulnerable to blind, short-time-scale replay attacks when the host sends consecutive identical packets in hopes that none of his packets will be selected, we make the following modest recommendations for mitigating the effect of replay attacks over long time scales, while still preserving the passive properties of the sampling scheme: (1) export packets should be secured (either by (a) collocating the Sampler and Collector so that export packets need not traverse the network, or by (b) encrypting, padding to constant length, and sending export packets over the network at constant rate), and (2) the PRF key should be changed (using 'forward secrecy' techniques, as in IKE [7]) to limit the time the time window of vulnerability to replay attacks, *e.g.* each time billing information is sent to the end hosts.

We believe that these modest recommendations could readily be deployed in secure implementations of the PSAMP charter.

Acknowledgments

The authors thank David Xiao, Boaz Barak, Haakon Ringberg and the members of the Cabernet Group at Princeton University for helpful comments and discussions, and Shai Halevi for help with the attack on the hash function $f_{a,b,k}(d) = ad + b \bmod k$. We also thank Nick Duffield for his insightful comments and questions, and for his work on incorporating our recommendations into the latest draft of IETF PSAMP charter.

References

[1] IETF, "Packet sampling working group," *http://www.ietf.org/html.charters/psamp-charter.html*.

[2] N. G. Duffield and M. Grossglauser, "Trajectory sampling for direct traffic observation," *IEEE/ACM Transactions on Networking*, vol. 9, no. 3, pp. 280 – 292, June 2001.

[3] N. Duffield, Ed., *A Framework for Packet Selection and Reporting, Internet Draft*. IETF Packet Sampling Working Group, January 2005 - work in progress.

[4] T. Zseby, M. Molina, N. Duffield, S. Niccolini, and F. Raspall, *Sampling and Filtering Techniques for IP Packet Selection, Internet Draft*. IETF Packet Sampling Working Group, July 2005 - work in progress.

[5] M. Bellare, J. Kilian, and P. Rogaway, "The security of the cipher block chaining message authentication code," *Advances in Cryptology - Crypto '94*, 1994.

[6] M. Bellare, R. Canetti, and H. Krawczyk, "Pseudorandom functions revisited: The cascade construction and its concrete security," *Symposium on Foundations of Computer Science*, 1996.

[7] D. Harkins and D. Carrel, *Internet RFC 2409: The Internet Key Exchange (IKE)*, 1998.

[8] M. Bellare and P. Rogaway, *Random oracles are practical: a paradigm for designing efficient protocols*. ACM First Annual Conference on Computer and Communications Security, 1993.

[9] NIST, "Hash function workshop," *http://csrc.nist.gov/pki/HashWorkshop/*.

[10] D. Boneh and R. Venkatesan, "Hardness of computing the most significant bits of secret keys in diffie-hellman and related schemes," *Proceedings Crypto '96*, vol. 1109, pp. 129–142, 1996.

[11] W. Alexi, B. Chor, O. Goldreich, and C. P. Schnorr, "RSA and Rabin functions: certain parts are as hard as the whole," *SIAM J. Comput.*, vol. 17, no. 2, pp. 194–209, 1988.

[12] O. Goldreich, *Foundations of Cryptography*. Cambridge University Press, 2007.

[13] S. Goldberg, D. Xiao, B. Barak, and J. Rexford, "Measuring path quality in the presence of adversaries: The role of cryptography in network accountability," *Princeton University, Department of Computer Science, Technical Report*, May 2007.