Strange Attractors and TCP/IP Sequence Number Analysis

Author: Michal Zalewski <lcamtuf@bos.bindview.com>

Printer-friendly version of this paper can be downloaded here.

Table of Contents:

0. Abstract

We consider the problem of inserting a malicious packet into a TCP connection, as well as establishing a TCP connection using an address that is legitimately used by another machine. We introduce the notion of a Spoofing Set as a way of describing a generalized attack methodology. We also discuss a method of constructing Spoofing Sets that is based on Phase Space Analysis and the presence of function attractors. We review the major network operating systems relative to this attack. The goal of this document is to suggest a way of measuring relative network-based sequence number generators quality, which can be used to estimate attack feasibility and analyze underlying PRNG function behavior. This approach can be applied to TCP/IP protocol sequence numbers, DNS query identifiers,

session-id generation algorithms in cookie-based authentication schemes, etc.

Please note that presented results are preliminary and should not be considered as a reliable metric for comparing the relative strength of the operating systems ISN generators at this time.

1. Introduction

1.1 TCP Sequence generation and PRNGs

Upon connection via TCP/IP to a host, the host generates an Initial Sequence Number (ISN). This sequence number is used in the conversation between itself and the host to help keep track of each packet and to ensure that the conversation continues properly. Both the host and the client generate and use these sequence numbers in TCP connections.

As early as 1985 there was speculation that by being able to guess the next ISN, an attacker could forge a one-way connection to a host by spoofing the source IP address of a trusted host, as well as the ISN which would normally be sent back to the trusted host via an acknowledgement packet. It was determined that to help ensure the integrity of TCP/IP connections, every stream should be assigned a unique, random sequence number. The TCP sequence number field is able to hold a 32-bit value, and 31-bit is recommended for use by RFC specifications. An attacker wanting to establish connection originating from a fake address, or to compromise existing TCP connection integrity by inserting malicious data into the stream [1] would have to know the ISN. Because of the open nature of the Internet, and because of large number of protocols that are not using cryptographic mechanisms to protect data integrity, it is important to design TCP/IP implementations in a way that does not allow remote attackers to predict an ISN (this is called a "blind spoofing" attack).

It is difficult to generate unpredictable numbers using a computer. This is because computers are designed to execute strictly defined sets of commands in repeatable and accurate ways. Thus, every fixed algorithm can be used to produce exactly the same results on another computer, which then can be used to effectively predict output values, assuming attracker can reconstruct internal state of such a remote system. Also, even if the target PNRG function is not known, sooner or later the algorithm will start generating the same exact sequences over again, because there is a limited number of possible internal states that can be used by a specific algorithm (computers are using finite precision and range arithmetics). Hopefully this happens later and the conditions to start the repeating of sequential numbers will take many months or years. But, there are known vulnerable implementations with a PRNG generator period of just over 500 elements or less.

The common approach of dealing with this lack of true randomness is to introduce additional randomness, or entropy, from an external, unpredictable source. Usually, this randomness is calculated from keystroke intervals, specific I/O interrupts and other parameters that are not known to the attacker. This solution, combined with a reasonably good hashing function that produces full 32 or 31-bit data with no correlation between subsequent results without revealing useful information about the internal state of PRNG function, can be used to make an excellent TCP sequence generator. Unfortunately, TCP ISN generators are rarely written this way, and when they are, there are numerous flaws or implementation errors that can lead to predictable ISNs.

RFC1948 suggests the use of source IP address, destination IP address, source port and destination port, plus an additional random secret key. This data should be hashed using a shortcut function to generate random and unique sequence numbers for every unique connection. Failing to account for this can lead to improper conclusions when analyzing TCP generators with respect to ISN predictability. Indeed, statements that are true for the ISNs coming back to the attacker might not be true for other connections, as the hash values would be different.

This research attempts to analyze the pseudo-random number generators (PRNGs) used for TCP sequence number generation in different operating systems and to expose potential flaws in the algorithms used. We analyzed the generated sequence numbers, instead of trying to focus on the actual implementations in the various operating systems. In essence, we approached the analysis from

the same standpoint as the remote attacker would - from the network.

[1] - Data insertion attacks require additional knowledge about established connections, which can be easily obtained in some cases. However, no knowledge is required about the transferred data itself so, these attacks qualify as "blind spoofing" attacks. Such attacks are performed against open client-server connections based on the knowledge about predictable ISNs used at one of the endpoints, and can be performed against numerous protocols. One of the examples is inserting malicious contents or malicious RCPT TO fields into SMTP transaction in order to modify or intercept e-mails.

1.2 Spoofing Sets

TCP implementations must be reasonably robust against denial of service (DoS) attacks. Among other things, this means that all TCP implementations regularly discard packets with incorrect ISNs, since the failure to do so presents an obvious DoS attack. TCP ISNs are 32-bit numbers. So, if an attacker is able to generate 2^32 packets, each with a different guess for the next sequence number, the attacker would be assured that one of his malicious packets will contain the correct sequence number. However, guessing the right ISN from the entire 32-bit space (4,294,967,296 possibilities) is not feasible due to the excessive amount of bandwidth and time required. That is why a good TCP sequence number generator implementation currently provides enough security to protect against spoofing attacks, at least for the present time and in typical conditions. But increasing bandwidth and processor speed will eventually make brute force guessing of 32-bit ISNs feasible for the average attacker.

Based on these observations, we introduce the idea of a Spoofing Set. A Spoofing Set is a set of guessed values for ISNs that are used to construct a packet flood that is intended to corrupt some established TCP connections. Our goal is to add enough reasonable guesses to the Spoofing Set to ensure that the next ISN value is included, while at the same time, keeping the Spoofing Set size small enough for an attack to be feasible. A Spoofing Set can be as small as a single value or it as large as a several million posibilities.

Attacks with less than 5,000 combinations are usually feasible regardless of network uplink parameters and available resources. Attacks with 5,000 to 60,000 possibilities are more resource-consuming, but still possible. Attacks using more than 60,000 packets are possible only rarely and would consume amounts of bandwidth and resources that are not available to all attackers.

2 Introduction to Phase Space Analysis

In this section we provide a short introduction to phase space analysis and the behavior of strange attractors. We then discuss how to apply these tools to the problem of ISN value guessing.

2.1 Introduction to Phase Space Analysis

Phase space is an n-dimensional space that fully describes the state of an n-variable system. An attractor is a shape that is specific to the given PRNG function, and reveals the complex nature of dependencies between subsequent results generated by the implementation.

We wanted to generate clean, three-dimensional representations of one-dimensional input data. The method used is known as "delayed coordinates", and is well-known and widely used in the analysis of dynamic systems, especially nonlinear systems and deterministic chaos [2]. This method assumes that we can reconstruct missing dimensions using previous, delayed function values as additional coordinates. Instead of using function values, we decided to calculate the first-order difference for the input data to generate more suggestive and useful results to show the function dynamics. So if s stands for the input set, and x, y and z are the point coordinates we are looking for, the equations are:

```
x[n] = s[n-2] - s[n-3]
y[n] = s[n-1] - s[n-2]
z[n] = s[n] - s [n-1]
```

The following is an example of input data from a sequence of ISNs. Looking at the example doesn't

allow us to determine what kind of underlying function was used to generate the data. It appears that these numbers are random with no dependencies between subsequent results:

...4293832719, 3994503850, 4294386178, 134819, 4294768138 191541, 4294445483, 4294608504, 4288751770, 88040492...

Here is what we would see when using the attractor reconstruction technique:

What we see is an order and a correlation between subsequent results, creating a subtle three-dimensional path.

2.2 Using Attractors for Spoofing Set Construction

We can now describe how to use 3-dimensional PRNG function attractors to construct a Spoofing Set. Since the behavior of attractors is not fully understood, we note that the algorithm presented here is a heuristic approach. We can not prove, in a strict mathematical sense, that our algorithm will accurately guess ISN values. Nor have we done the statistical analysis that would be required to verify that our results are statistically significant and predictive of future results. Such analysis would require the collection of more data and is beyond the scope of this paper. We hope this sort of independent verification will be an area of future work.

Our approach is built upon this widely accepted observation about attractors:

If a sequence exhibits strong attractor behavior, then future values in the sequence will be close to the values used to construct previous points in the attractor.

Our goal is to construct a spoofing set, and, later, to calculate its relative quality by empirically calculating the probability of making the correct ISN prediction against our test data. For the purpose of ISN generatorscomparison , we established a limit of guess set size at the level of 5,000 elements, which is considered a limit for trivial attacks that does not require excessive network bandwidth or processing power and can be conducted within few seconds.

We assume the targeted system's operating system is known and that we have already collected a sample sequence of approximately 50,000 ISN values.

We will call this sequence seq[n], and will call the corresponding set of 3-dimensional phase-points calculated based on the deltas between sequence values, A. Remember, the coordinates for points in the attractor, A, are calculated using the equations:

```
x[t] = seq[t]   - seq[t-1]
y[t] = seq[t-1] - seq[t-2]
z[t] = seq[t-2] - seq[t-3]
```

Here is an example of the 3-dimensional attractor for some sequence, seq[n]:

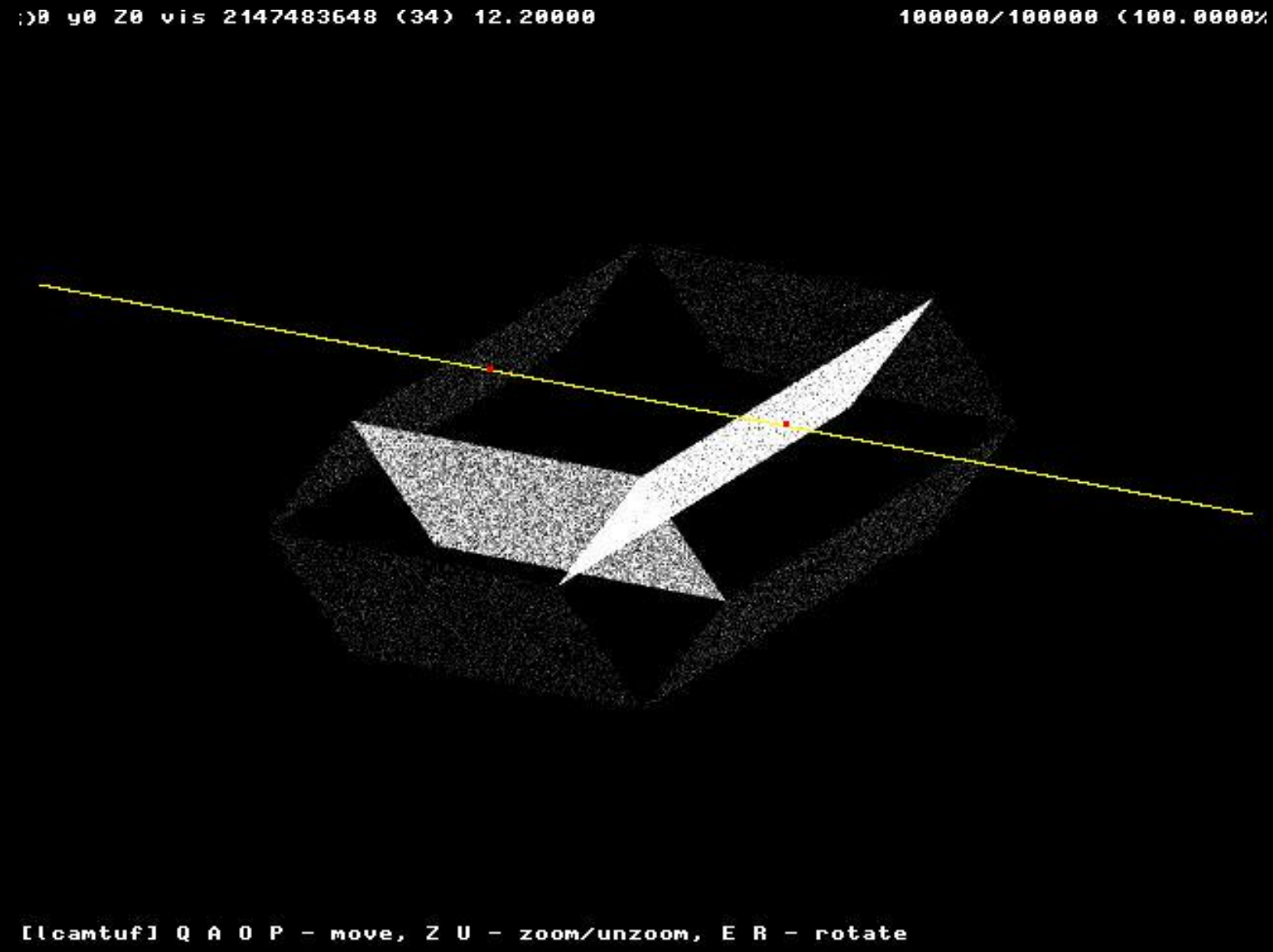
If we know the value of seq[t-1], the problem of determining a "good" guess for the value of seq[t] is equivalant to choosing a "good" point in discrete 3-space. Indeed, given a point (x, y, z), we can add x + seq[t-1] to our Spoofing Set as a guess for seq[t].

Now, we turn our attention towards choosing points in discrete 3-space that will produce effective Spoofing Sets. We refer to adding a point (x, y, z), adding a delta value x and adding a sequence value to the Spoofing Set interchangeably.

We begin this analysis by noting that seq[t-1], seq[t-2] and seq[t-3] can be easily gathered by probing the remote host. Thus, the point (x[t], y[t], z[t]), which corresponds to the next sequence value, seq[t], will be somewhere on the line, L, given by:

```
y = seq[t-1] - seq[t-2]
z = seq[t-2] - seq[t-3]
```

If the effect of the 3-D attractor is strong, then it is reasonable to assume that the actual value of seq[t] will correspond to a point that is in, or is close to, the intersection of L and A.



So, we now consider how to build up our Spoofing Set. It is useful to think of this as a three phase process. First, we want to include any points in the intersection between our line, L, and the attractor, A. However, we note that this intersection may be empty; a situation that occurs if the subsequence seq[t-3], seq[t-2], seq[t-1] never appears in our original input data stream. In this case, we want to add points in A that are "close" to our line, L.

To explain this approach, we will describe how to construct a Spoofing Set using a hypothetical example. Let's assume that the three previous sequence values were equal to 5, 10 and 7816. Further suppose we do not have any point in the attractor matching the y and z coordinates calculated using these deltas (y = 10-5 = 5, z = 7816-10 = 7806), but there is another attractor point, for y = 6 and z = 7805. If the PRNG is not working properly and there is some kind of correlation between the previous and the next results, we can expect that points with almost the same y and z coordinates would have almost the same x coordinates. Of course, this may not be true, but if it is, then it is a clear sign of a weak PRNG. Assuming it's true, we want to include this point in our Spoofing Set, and hope that the next ISN will be relatively similar.

So, the second phase of our process is to select all points in A that are within some radius, R1, of the line, L, and use their corresponding x values to create candidates for the Spoofing Set.

Finally, the observed behavior of strong attractors is that their shape tends to fill up and become more dense as more points are plotted. This implies that the value of x(t) should be relatively close to the x-value of one of the points already in our Spoofing Set.

Geometrically speaking, we can think of taking the projection of all points currently in our Spoofing Set onto line L or in near proximity of it (radius R1). Then for each point in the projection we create an interval with radius R2, and add each of the integer values within these intervals to the Spoofing Set. More precisely, for each value x in the initial Spoofing Set, we add the following values:

```
x + 1, x - 1, x + 2, x - 2, ... x + R2, x - R2
```

Because we want to implement a fixed limit of 5,000 elements per guess set, we have to select R1 in the way that would generate non-empty guess sets (usually of the size 1-500) in the phase 2, and choose R2 to produce exactly 5,000 elements (excluding possible overlaps in generated ranges for every ISN guess obtained in phase 2).

We have produced a program that implements this algorithm, and performs an attractor space search to predict the next ISN values.

The program accepts four parameters. These parameters are the three previous sequence numbers and the search radius R1, which affects search speed and does not have significant meaning for the generated results as long as any points can be found. The program tries to guess the next ISN. The ISN guesses generated correspond to the values that are used to generate attractor points. For more precise results, the Spoofing Set can be sorted first by the "attraction strength" (point saturation) of every point and then, for points of similar attraction strength, by distance from the given y and z line.

Separate program was used to calculate the value of R2 that has to be used for the initial Spoofing Set returned previously in order to achieve Spoofing Set of size exactly 5000.

For PRNG function analysis, we collected streams of 100,000 sequence numbers for many popular operating systems. For each operating system, we divided our input data set into two parts. The first 50,000 were used by our algorithm to reconstruct the phase-space attractor. Then, trials were conducted to analyze relative Spoofing Set quality. This was done by inspecting all possible quadruples of ISNs from the second 50,000 and calculating the number of values that could be predicted using the Spoofing Set generation algorithm described above. The percentage of successful guesses is referred to as the "attack feasibility ratio".

Additional spoofing set characteristics are calculated at the time of initial tests:

- Average R2 radius, which reflects attractor strength; a larger R2 radius indicates a stronger attractor structure while a smaller R2 indicates a more dispersed structure

- Average error, which reflects average distance from the numbers generated in stage 2; it is always smaller than R2, and a large disproportion means that usually less than 5,000 guesses has to be made for successful ISN prediction,

- Average number of elements generated in stage 2, and average number of elements to be processed before getting the right guess; this, for given R1, reflects attractor density,

Note that average error value can be calculated only if there is enough correct guesses.

These values, along with the attack feasibility and specific attractor 3D representation, are unique for every operating system or ISN implementation.

To minimize or avoid the risk caused by the RFC1948 suggestions mentioned earlier, our test data was generated using a constantly changing source port number. For properly working hashing functions built according to the RFC specifications, changes of used source port would cause effects that are not

distinguishable from changes caused by different source addresses. Other behaviors would mean that the hashing function could be easily reversed and is not secure for PRNG or RFC1948 purposes. Whenever RFC1948-specific behavior was observed, additional tests with constantly changing source address IPs were performed to confirm this assumption.

Note: Our test set of approximately 50,000 quadruples is not a true random sampling of real-life data. The quadruples are subsequent to each other and are subsequent to the data set used to reconstruct the attractor. For this reason, we must point out that our coverage rate can not be interpreted as being predictive of future success. It should be relatively straight forward to perform the requisite statistical analysis to be able to make statements about the accuracy of our initial trials, but this is beyond the scope of this paper.

2.3 Real-Life Attack Algorithm

The results presented in this document were produced using relatively good network parameters on LAN networks or fast Internet uplinks. For some algorithms, mainly time-dependent algorithms, the generated attractors are specific for given packet latency ranges. For example, an attractor built for a network where SYN packets (and SYN+ACK responses) are delivered in the intervals of 10 to 20 milliseconds would be unsuitable or would produce significantly lower quality guesses against a host with packet delivery intervals of 500-800 ms.

Because of this, it may be a good idea to generate attractors for few common timing ranges for each operating system, and use the best for a specific attack. Nevertheless, unstable or overloaded links, where average delivery delays vary heavily (e.g. 10-5000ms), might decrease the relative quality of Spoofing Sets generated against time-dependent algorithms. This observation does not apply to other algorithms that show no time-dependency patterns. Additionally, excessive packet drop ratios and other network conditions can affect attack feasibility.

For the purpose of estimating attack feasibility and choosing the best attractor for a given operating system, applied patches and network latency, we recommend the following algorithm:

A) The attacker has a set of attractors built for specific systems (and in specific network conditions). Every attractor should be described with additional parameters, that include estimated attack feasibility, average R2, average error, average elements count, etc.

B) The attacker should get n data samples, each containing four subsequent sequence numbers from the host to be attacked. The operating system of the target does not need to be known at this point. In most cases, n=20 should be sufficient. This it requires sending 80 packets.

C) For each attractor in the archive, the attacker should use the first three sequence numbers in each data sample to predict the fourth, by using the algorithm discussed above and the R1 that belongs to this attractor. The attacker should choose the attractor with the best results.

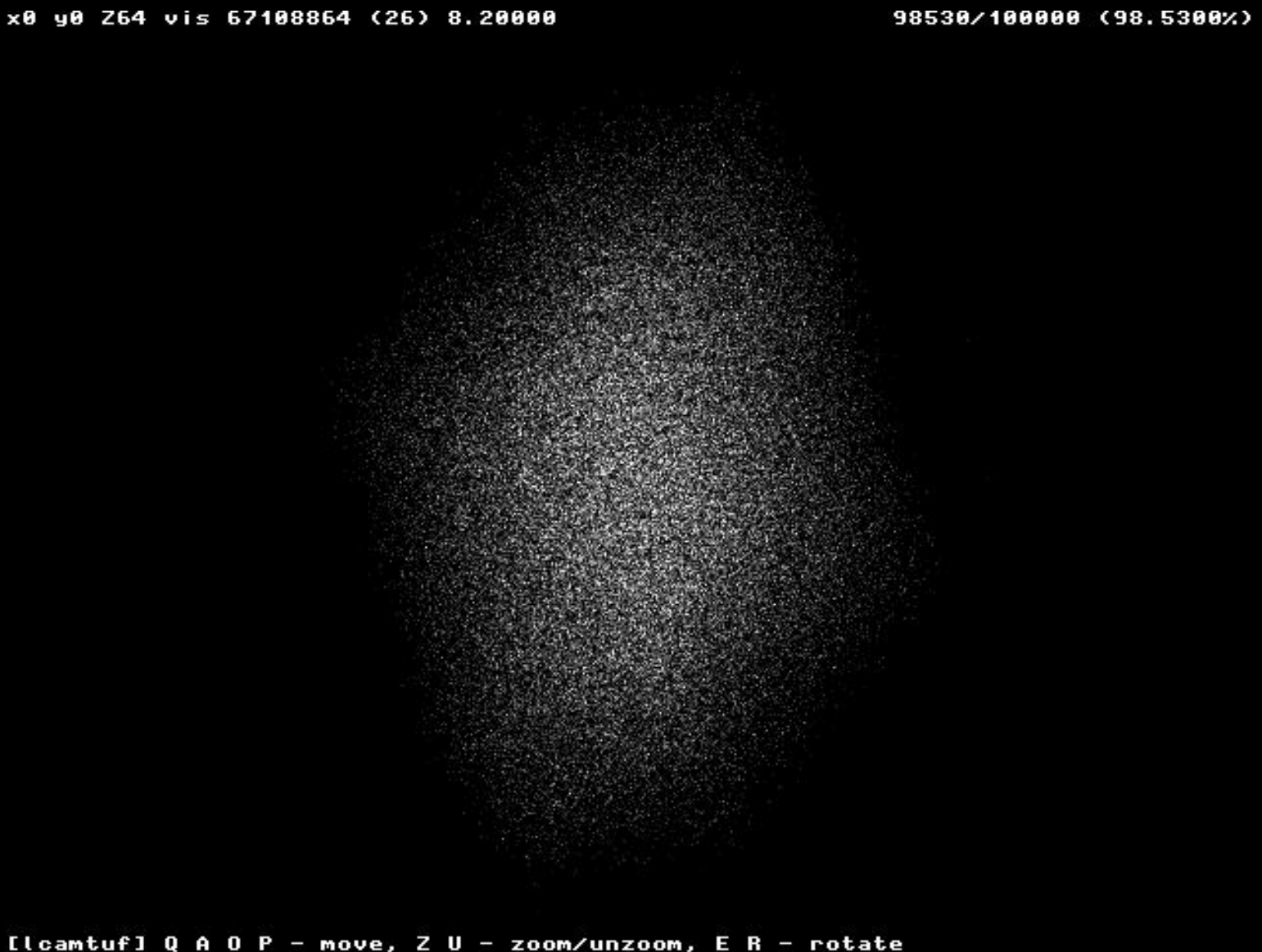
D) Then, the attacker has to wait until the attack condition happens. The attacker can then collect the three sequence numbers, create a Spoofing Set with the selected attractor and perform the attack.

E) If no suitable attractor has been found, the specific operating system or specific network parameters cannot be exploited in a feasible way using the known data, then here are two ways to solve this problem. First, the attacker can use target itself to build an attractor. This method would be very accurate, but it involves sending thousands of packets, which might be noticed. Or, they can perform operating system fingerprinting, and reproduce the targeted system configuration in a lab or locate it somewhere else, where attack would not be noticed. Then, the attacker needs to measure the average network latency and the fluctuations, and reproduce the conditions during the attractor construction process. Finally, standard attractor test procedures (probability estimations, choosing R1, etc.) need to be performed.

3. Review of Operating Systems

In this section, we discuss the specific TCP/IP ISN generators used by a variety of operating systems.

3.1 Linux



Operating system:	Linux 2.2
R1 radius:	1000
Attack feasibility:	below 0.05%
Avg. number of elements:	27 / n/a
Average R2:	1415
Average error:	n/a

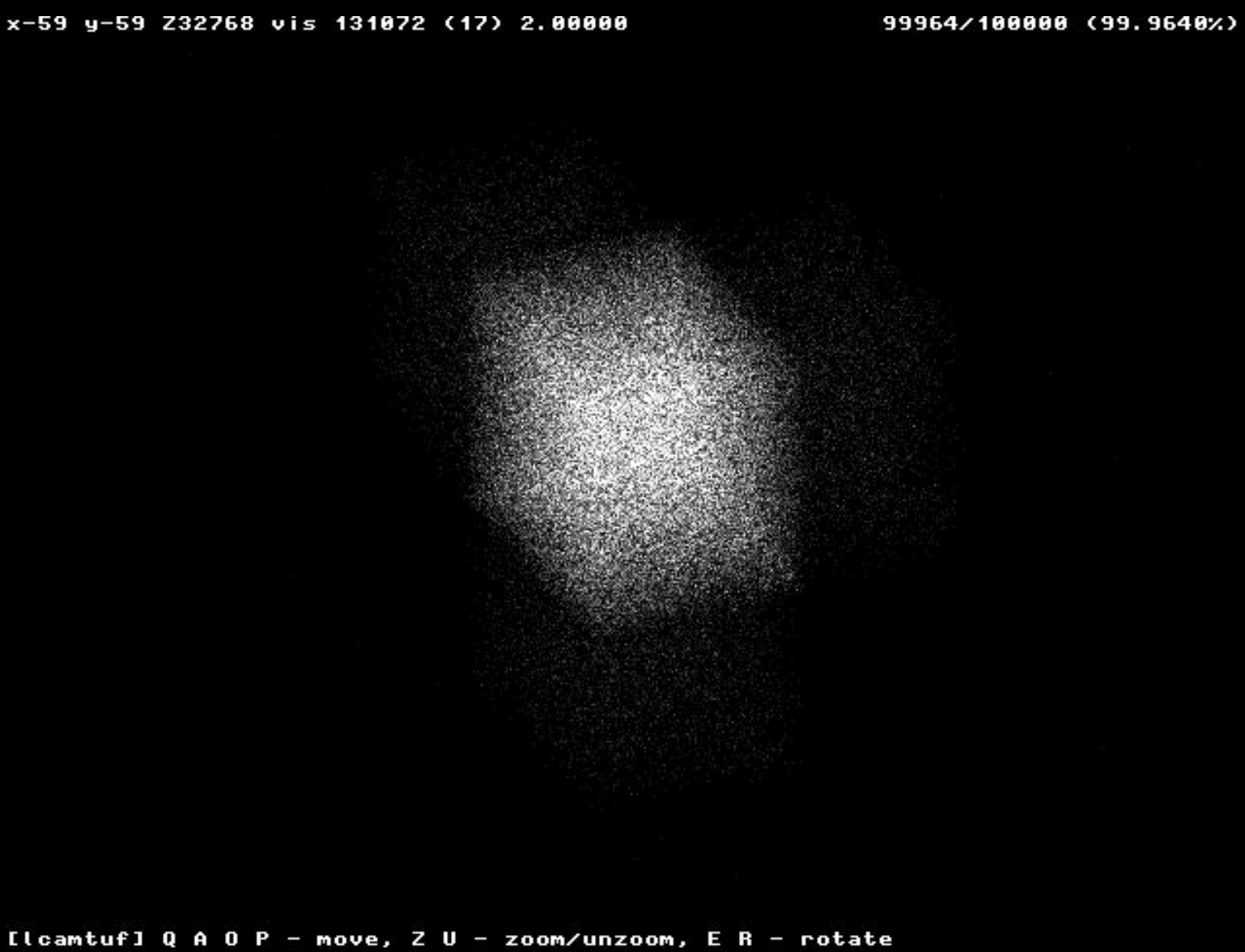
Linux 2.2 TCP/IP sequence numbers are not as good as they might be, but are certainly adequate, and attack feasibility is very low. There is no strong attractor structure; it makes a 24-bit wide cloud, which means that the deltas are in the range C to C + 2^b - 1, where C is a constant shift and b is a bit width of the cloud. A wider 32- or 31-bit cloud would be better, but 24 bits gives over 16 million combinations, making spoofing practically impossible in most real-life scenarios.

Our attractor analysis method gives results comparable to a brute-force search of whole 24-bit space. Linux can be qualified as low-risk OS in the scope of this document.

Linux is taking advantage of RFC1948. The hashing function implementation seems to be flawless, and additional randomness is introduced. Thus, the observed ISN generator characteristics are not related

to any specific TCP connection parameters. ISNs are more easily predictable when using exactly the same source port, source address, destination port and destination address, but hashing function "secret" value is modified in relatively short time intervals (and thus frequently changing observed characteristics), do not exposing system security.

3.2 Windows



Operating system:	Windows 2000
R1 radius:	10
Attack feasibility:	12.08%
Avg. number of elements:	81 / 5
Average R2:	413
Average error:	151

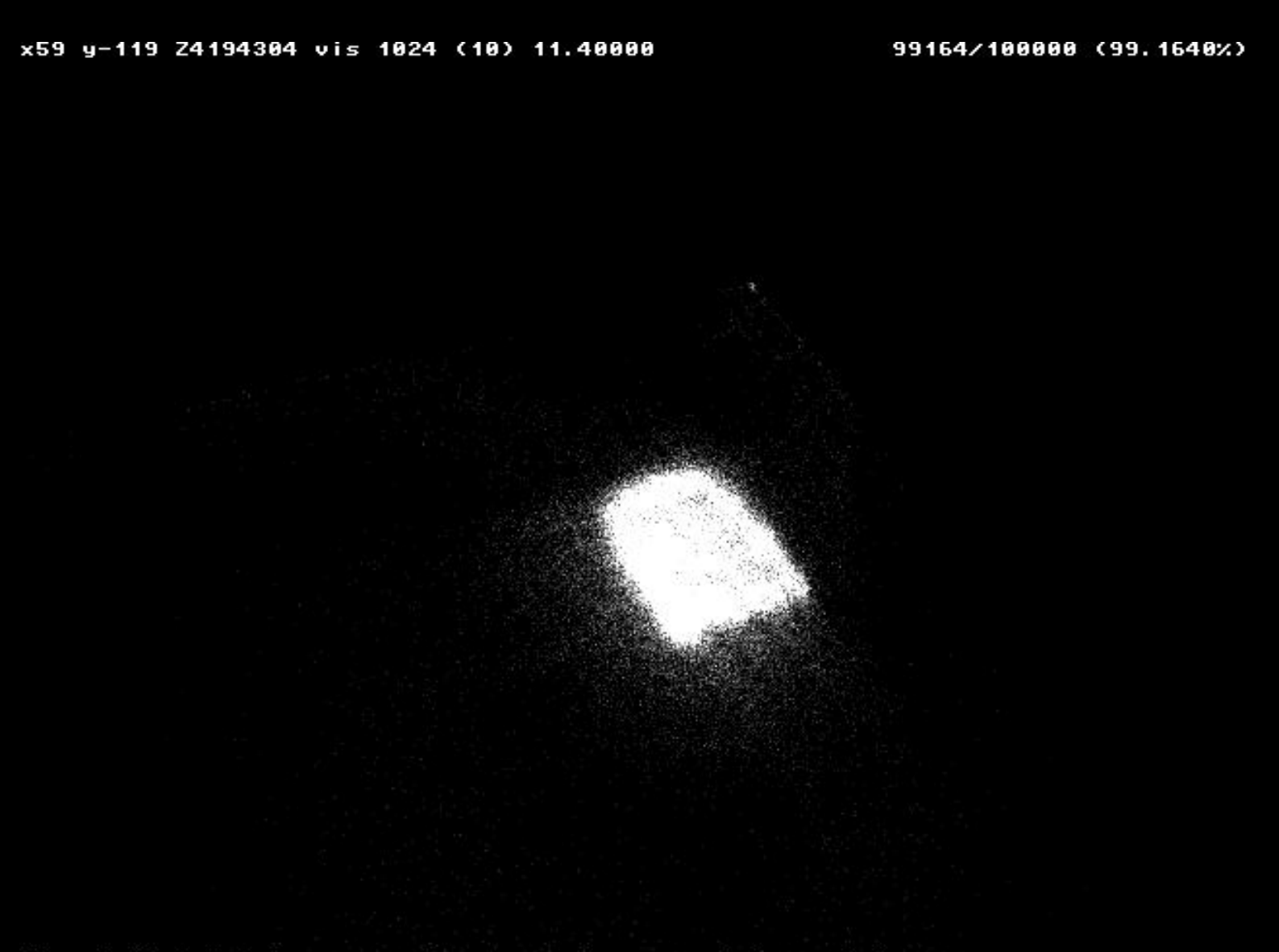
Operating system:	Windows NT4 SP6a + hotfixes
R1 radius:	10
Attack feasibility:	15%
Avg. number of elements:	106 / 5
Average R2:	426
Average error:	177

Windows 2000 and Windows NT4 SP6a are presenting almost the same level of TCP sequence

number predictability, which can be qualified as medium to high, allowing attacker to get reasonably high success rates without using excessive amounts of network resources. Both systems are mildly vulnerable to attacks. There is no strong attractor structure visible (3D "cube").

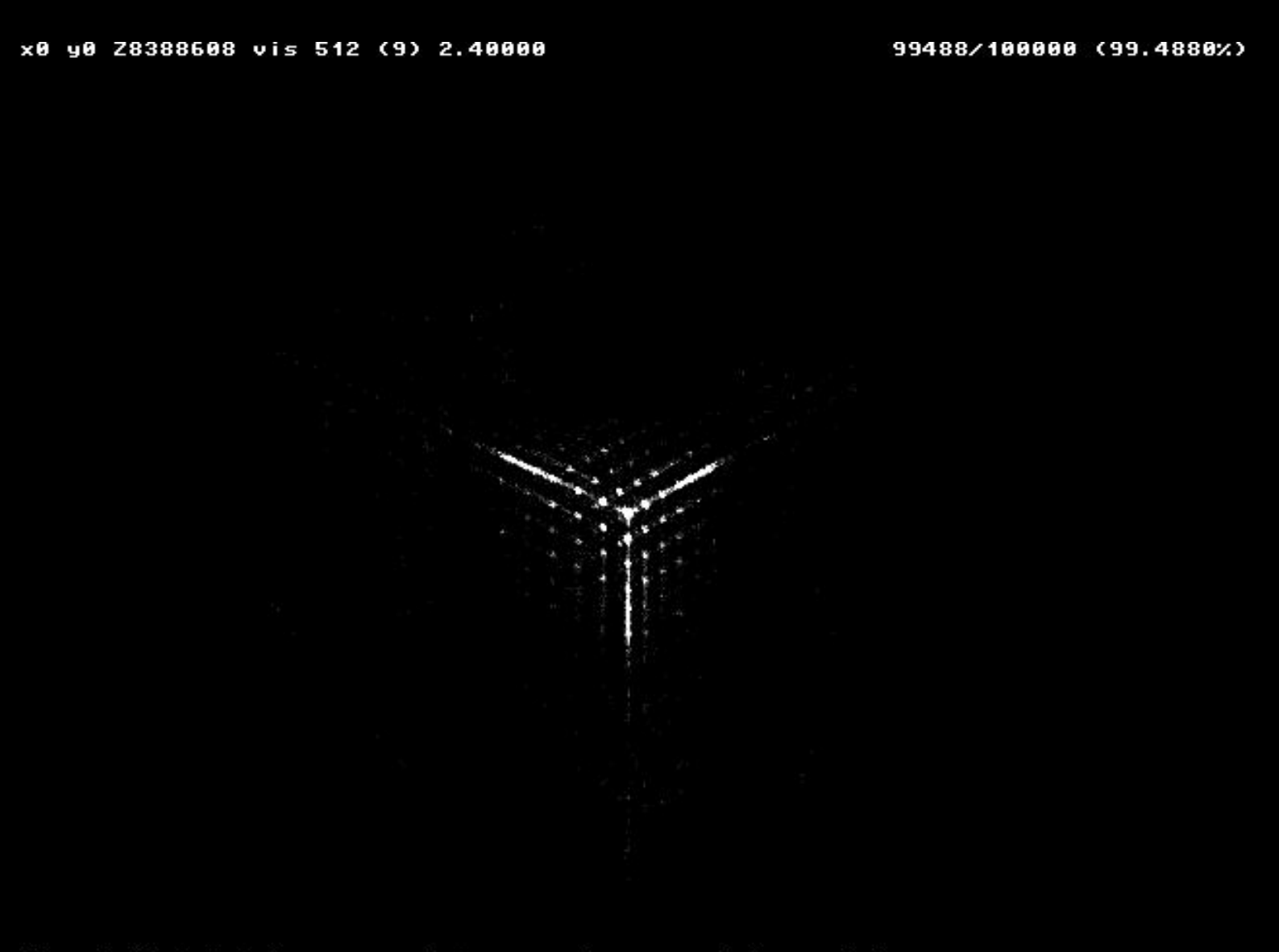
Operating system:	Windows NT4 SP3
R1 radius:	0
Attack feasibility:	97.00%
Avg. number of elements:	570 / 2
Average R2:	670
Average error:	8

Windows NT4 with no recent service patches and hotfixes is vulnerable to ISN guessing attacks using just a few packets, and can be easily attacked using 5,000 guesses with an almost 100% success rate. This version of Windows NT 4.0 can be qualified as high risk. Our attractor analysis method gives very good results here. NOTE: Problems in pre-SP6a Windows NT were already addressed in several advisories.



Operating system:	Windows 95
R1 radius:	0
Attack feasibility:	100.00%
Avg. number of elements:	1048 / 1
Average R2:	2294
Average error:	118

Windows 95 sequence numbers are very weak. But it is really difficult to understand is why this algorithm was further "weakened" in Windows 98 (SE), decreasing estimated error and number of elements required to get the right guess, in average:



Operating system:	Windows 98 SE
R1 radius:	0
Attack feasibility:	100.00%
Avg. number of elements:	785 / 1
Average R2:	1091
Average error:	7

3.3 Cisco IOS

Recently, some security advisories addressing serious TCP/IP sequence number flaws in the Cisco IOS operating system were released. Short tests against Cisco IOS implementation show the nature of this vulnerability:

Operating system:	Cisco IOS 12.0 (unpatched)
R1 radius:	10
Attack feasibility:	20.00%
Avg. number of elements:	88 / 3
Average R2:	557
Average error:	431

What we see above is a trivial, probably microsecond clock-based time dependency at its finest (see Appendix A), with most of the points attracted to one point with "echos" around. In order to exploit this vulnerability, the attacker would simply have to synchronize his own clock with the IOS clock, taking care of packet rtt times, and performing the attack at any time by sending packets for, e.g., 100 microseconds +/-. Thus, Cisco IOS can be qualified as a high-risk OS. Besides that, our attractor analysis method provides reasonably high success ratio using 5,000 guesses against this OS.

Here, for comparsion, are the results for patched IOS. The main attractor is still there, but some additional noise was introduced to make the prediction more difficult:

Operating system:	Cisco IOS 12.0 (patched)
R1 radius:	1000
Attack feasibility:	2.06%
Avg. number of elements:	143 / 3
Average R2:	296
Average error:	182

3.4 AIX

This is a trivial cyclic increments example. The output doesn't look too impressive because there are just a few values used:

Operating system:	AIX 4.3
R1 radius:	0
Attack feasibility:	100.00%
Avg. number of elements:	99 / 1
Average R2:	278
Average error:	0

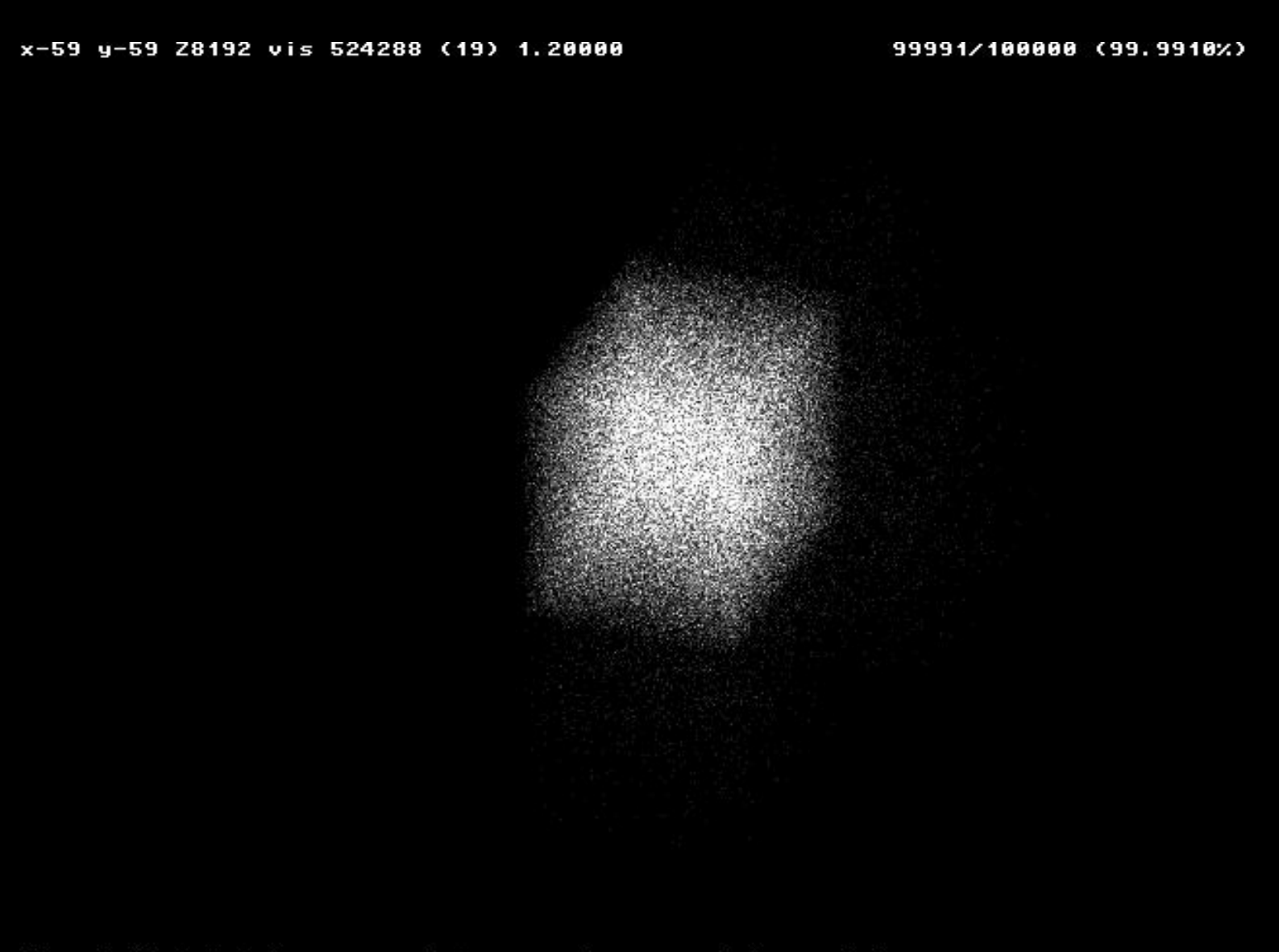
3.5 FreeBSD and NetBSD

FreeBSD 4.2 implementation is not impressive, and can be qualified as a medium to low risk system. An attack with minimal resources has a small, but non-zero success ratio:

Operating system:	FreeBSD 4.2
R1 radius:	10
Attack feasibility:	1.00%
Avg. number of elements:	59 / 2
Average R2:	129
Average error:	64

3.6 OpenBSD

The problem here is pretty similar to the FreeBSD case - not alarming, but not impressive. Tested against OpenBSD 2.8:



Operating system:	OpenBSD 2.8
R1 radius:	20
Attack feasibility:	3.05%
Avg. number of elements:	63 / 4
Average R2:	660
Average error:	372

The OpenBSD TCP/IP sequence number generator has recently been rewritten by Niels Provos. New code is available, but had not been included in any official release as of this writing. According to Theo de Raadt, the code was finished in December, and is supposed to be shipped with OpenBSD 2.9 in late May. The current version of OpenBSD generates a 31-bit wide cloud which does not produce any useful spoofing sets:

Operating system:	OpenBSD-current
R1 radius:	1000000
Attack feasibility:	0.00%
Avg. number of elements:	20 / n/a
Average R2:	1707
Average error:	n/a

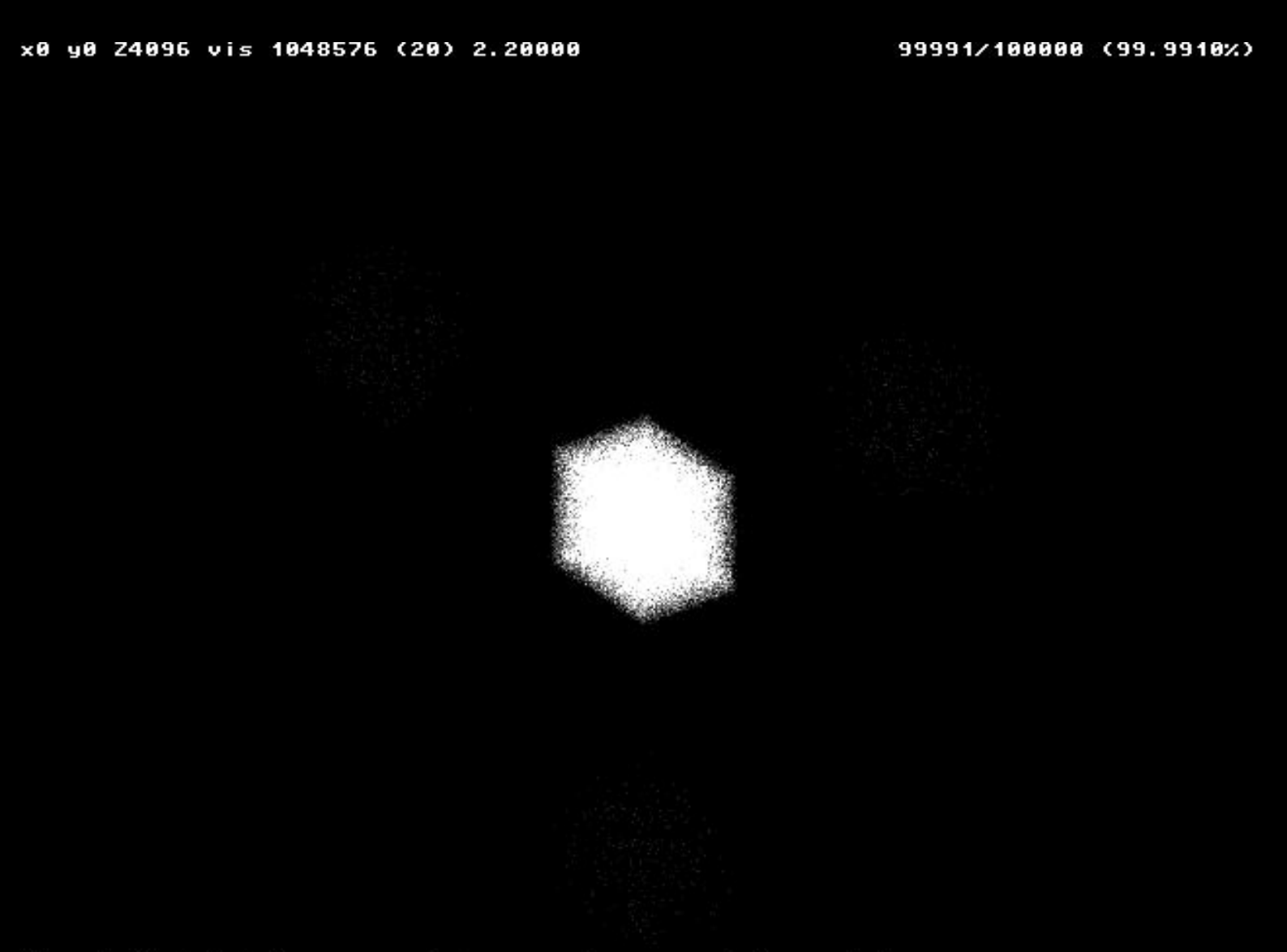
3.7 HP/UX

HPUX10 seems to have, practically speaking NO random sequence numbers. It has constant increases of 64,000 (0 bits of randomness, 1 guess, R1=0, R2=0). This is the default configuration, which fortunately can be modified, but once modified becomes a well-known cube pattern. HPUX11 seems to have a significantly improved random number generator. The function used seems to be really weird, reminding us of the shape of the Mir orbital station. Unfortunately, this is certainly not enough. It is possible to find a 12-element spoofing set, and HPUX can be described as a high risk system:

Operating system:	HPUX11
R1 radius:	0
Attack feasibility:	100.00%
Avg. number of elements:	10 / 1
Average R2:	2499
Average error:	0

3.8 Solaris

Solaris has three different settings for the tcp_strong_iss kernel parameter. When it is set to 0, completely predictable numbers are generated (a 9-element SpoofingSet). With the default setting of 1, Solaris 7 and 8 generates relatively good, but certainly not perfect, ISNs:



Operating system:	Solaris 7 (tcp_strong_iss=1)
R1 radius:	10
Attack feasibility:	2.08%
Avg. number of elements:	27 / 2
Average R2:	1292
Average error:	732

In Solaris 8 with tcp_strong_iss=1, the "randomness" source seems to behave slightly differently, but this does not affect its quality. The data makes oval, Linux-like cloud:

Setting tcp_strong_iss to 2, according to the vendor, is supposed to generate very reliable and unpredictable ISNs. The attractor generated for this operating system is filling 32-bit space rather uniformly. On the other hand, using the attractor analysis method, it is possible to guess the right value in less than 200 attempts, which can be explained by heavy saturation of attractor points:

Operating system:	Solaris 7 (tcp_strong_iss=2)
R1 radius:	100
Attack feasibility:	66.00%
Avg. number of elements:	19 / 1
Average R2:	2448
Average error:	108

It is important to note that, Solaris with tcp_strong_iss set to 2 seems to use the RFC1948 approach, which can be deduced by observing very low ISN deltas changes when using exactly the same source port and other TCP parameters in subsequent probes. To eliminate the risk described above when we discussed the issue of RFC1948 compliance, we repeated this tests for two different IP addresses, using the first data set to predict results found in the second. We found that the attractors generated in both cases are different. This does mean that you cannot use the data gathered using one IP address to perform attacks against another, because there is no consistency in ISN values when changing IP address.

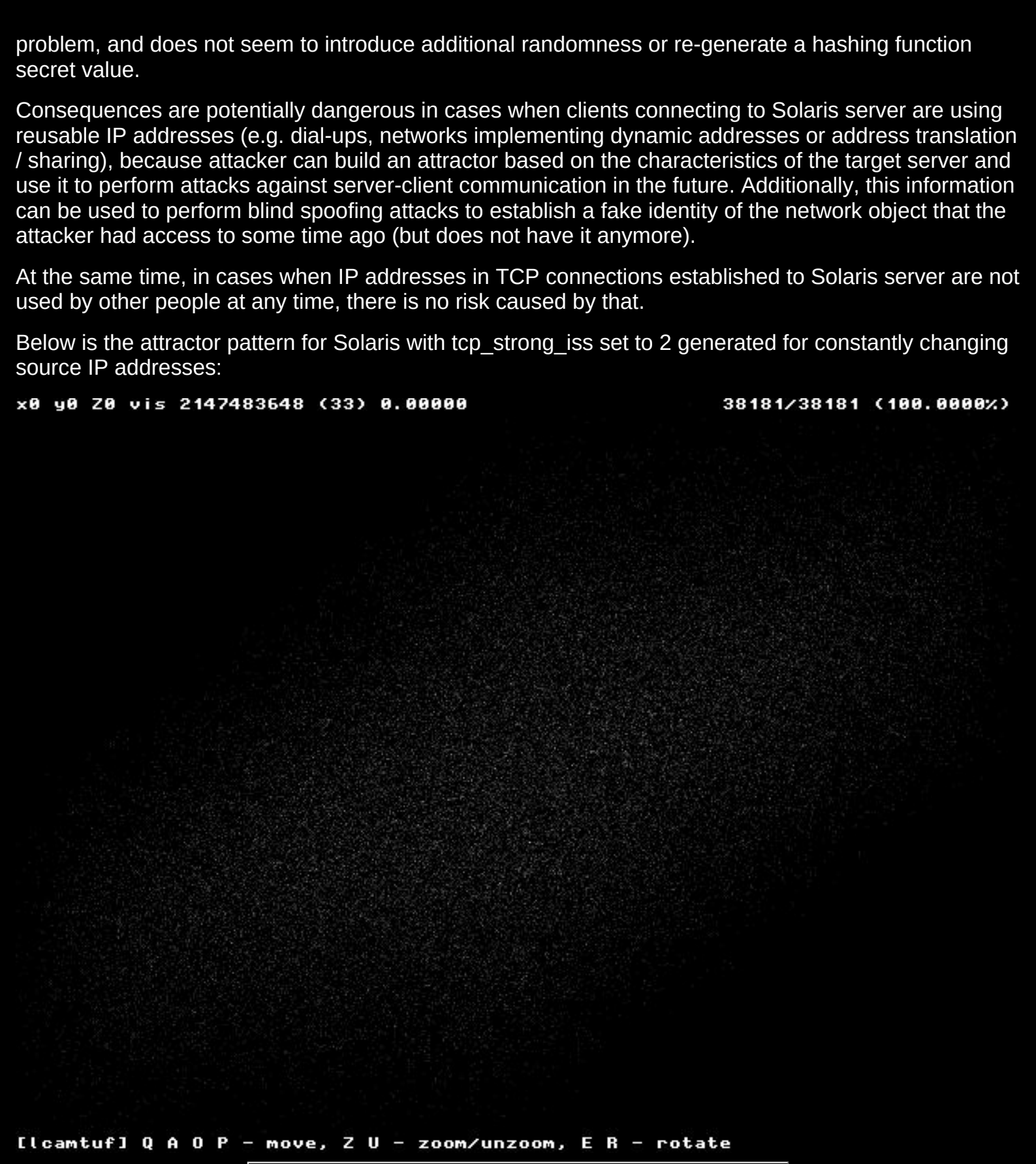
At the same time, this problem does not exist, for example, on Linux, which is implementing RFC1948 as well. This is because Linux is introducing additional randomness to ISNs, while Solaris does not, and generates constant, predictable patterns for the same IP address. The Solaris implementation is insufficient, because ISNs are almost completely dependent on shortcut function results, and for a few thousand of commonly used source port values, there could be no more than few thousand hashes (assuming other TCP connection parameters are not changing). Solaris does not compensate this

problem, and does not seem to introduce additional randomness or re-generate a hashing function secret value.

Consequences are potentially dangerous in cases when clients connecting to Solaris server are using reusable IP addresses (e.g. dial-ups, networks implementing dynamic addresses or address translation / sharing), because attacker can build an attractor based on the characteristics of the target server and use it to perform attacks against server-client communication in the future. Additionally, this information can be used to perform blind spoofing attacks to establish a fake identity of the network object that the attacker had access to some time ago (but does not have it anymore).

At the same time, in cases when IP addresses in TCP connections established to Solaris server are not used by other people at any time, there is no risk caused by that.

Below is the attractor pattern for Solaris with tcp_strong_iss set to 2 generated for constantly changing source IP addresses:



Operating system:	Solaris 7 (tcp_strong_iss=2)
R1 radius:	1000000
Attack feasibility:	0.00%
Avg. number of elements:	762 / n/a
Average R2:	208
Average error:	n/a

3.9 BSDI

Operating system:	BSDI 3.0
R1 radius:	10
Attack feasibility:	18.00%
Avg. number of elements:	70 / 3
Average R2:	779
Average error:	379

Both BSDI's 3.0 and 4.releases are similar to FreeBSD and give practically the same results, with a little bit more visible shadows. This makes the attack more difficult. The risk factor is medium to high.

3.10 IRIX

The IRIX operating system, even in versions that are meant to have relatively good ISN subsystem (recent 6.5 releases), seems to be flawed:

Operating system:	IRIX 6.5
R1 radius:	100
Attack feasibility:	20.00%
Avg. number of elements:	87 / 4
Average R2:	700
Average error:	200

This graph shows a 15-bit "flower" containing over 98% of all observed ISN deltas. Risk is medium to high.

3.11 MacOS

The older MacOS9 operating system has a predictable ISN generator. The output pattern is similar to an X-wing fighter:

Operating system:	MacOS 9
R1 radius:	100
Attack feasibility:	89.00%
Avg. number of elements:	1064 / 46
Average R2:	184
Average error:	30

The MacOS X operating system is another candidate for possible TCP sequence number guessing attacks, but is more secure than previous releases:

Operating system:	MacOS X
R1 radius:	10
Attack feasibility:	17.00%
Avg. number of elements:	121 / 6
Average R2:	384
Average error:	249

3.12 Multiple Network Devices

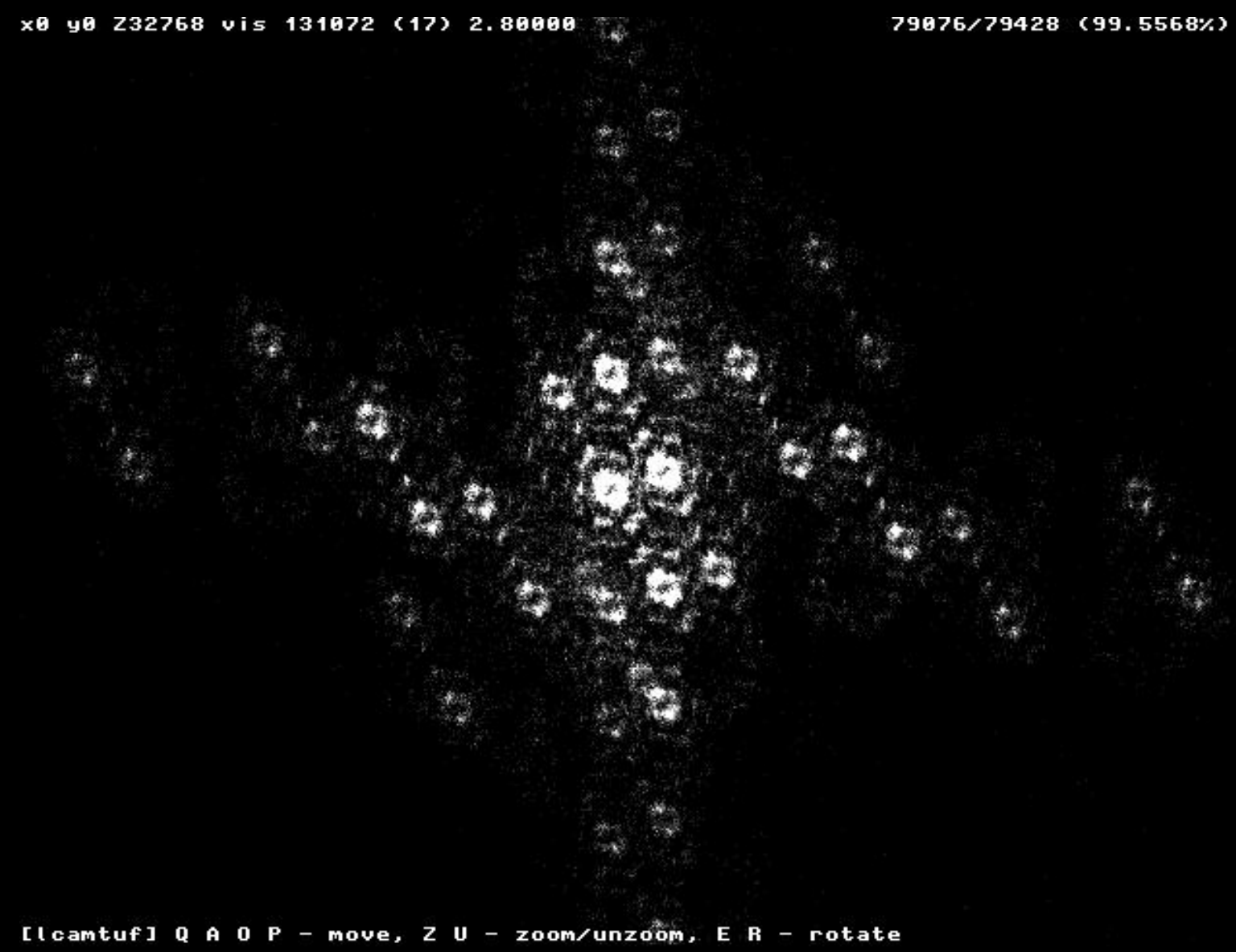
Most of the network devices like HP printers, many routers (Motorola, Netopia, US Robotics and Intel), Siemens IP phones, numerous 3Com switches and so on have completely predictable sequence numbers that use constant increments (or no increments at all). These devices would have a one-point or few-point representation of their PRNG engines. These problems are, in most cases, widely known and have been discussed on numerous forums (such as BugTraq), thus we do not think they are worth a separate discussion in this paper, but, as there is constantly increasing number of services provided by such a "smart" network equipment (for remote management, configuration or document delivery), this becomes more and more appealing problem.

3.13 Other PRNG issues

PRNGs are providing data source authentication not only in TCP protocol. Other uses are generating session-id "cookies" for secure web browsing and DNS protocol sequence numbers. DNS runs over UDP most of the time and the UDP protocol itself does not support sequence numbers. This paper is

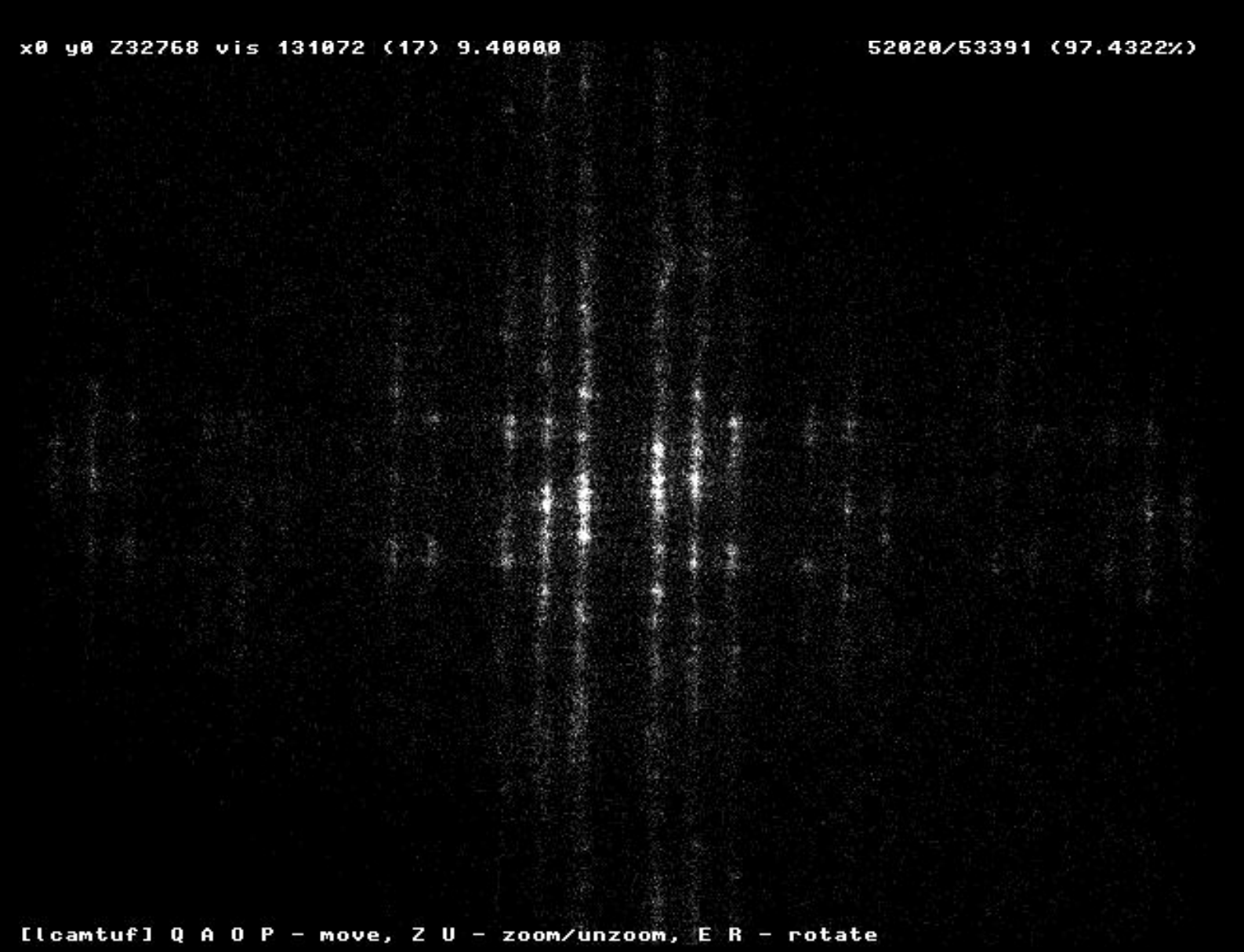
not going to discuss other PRNG applications in detail, but we would like to note the problem and demonstrate how easily our approach can be extended to, practically speaking, any [P]RNG implementation. First of all, three examples:

The following picture shows the weak glibc 2.1.9x resolver DNS sequence numbers implementation on Linux in client queries (over 50% attack feasibility, applying our previous algorithm to this case):



Below is the pattern generated for the Microsoft DNS server implementation (queries originating from the server), which shows very strong predictability pattern and is certainly not sufficient to protect against DNS spoofing (100% feasibility, average error of 2):

The last example is the Solaris 7 libc resolver pattern. It looks much more random than two examples mentioned above, but still have significant attractor patterns:

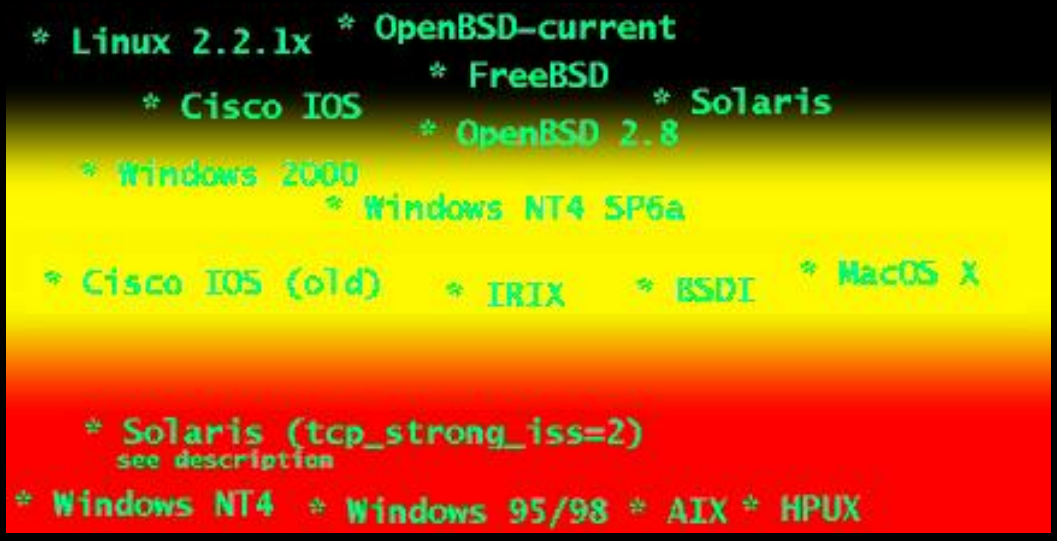


DNS sequence numbers are, generally speaking, weak, because only 16-bit space is used for this purpose. Weakening these numbers even more - as in MS DNS or Linux resolver case - allows attacker to perform almost instant DNS poisoning / DNS spoofing attacks. These attacks are even more dangerous, and certainly less complicated than TCP sequence number attacks. To intercept traffic originating from vulnerable client to, for example, www.microsoft.com server, attacker has to convince the DNS resolver implementation that this address should resolve to the attacker's IP address instead of the legitimate one. In case of caching servers or client-side daemons (like *nsd), this condition is even worse, because attack does not require human interaction and can be automated.

Recent bind releases and the FreeBSD resolver do not show attractor patterns. We have not tested another DNS daemon implementations (djbdns, etc) or resolvers (IRIX, HPUX, AIX), session-id generators in numerous other protocols, etc.

4. Risk Analysis

Below is the graph representing relative ranking of the operating systems tested in this survey:



OpenBSD-current (future 2.9 release) wins the competition. Linux 2.2.1x gets a relatively good score, but it is far from being perfect. Other systems generally received mediocre or very bad scores. Solaris with tcp_strong_iss set to 2 is interesting because it gets both one of the best and the one of the worst ranks at the same time, depending on whether or not the same Source IP is used to generate the attractor and to launch the attack.

We would like to note that using a strictly RFC1948 compliant implementation may not be a definitive answer to ISN guessing attacks. We make two observations. First, RFC1948 permits systems to create their ISN generation secret at boot time and to reuse this secret until shutdown. Second, IPv4 addresses are very often re-used (e.g. ISP dynamically assigned addresses, network address translations). Taken together these means that if an attacker were able to construct a scenario in which he could build an attractor (approx. 50k ISN numbers) with a set Source IP, then he could later launch an attack from that same Source IP with a reasonable chance of success, so long as the ISN generation secret on the target has not been recreated. This explains the large disparity between our Attack Feasibility estimates for Solaris with tcp_strong_iss=2 based on using the same Source IP (66%) and using different Source IPs (0%).

5. Conclusions

The general conclusions can seem a little scary. What comes to our attention is that most every implementation described above, except maybe current OpenBSD and Linux, has more or less serious flaws that make short-time TCP sequence number prediction attacks possible. Solaris 7 and 8 with tcp_strong_iss set to 2 results are a clear sign there are a lot of things to do for system vendors. We applied relatively loose measures, classifying attacks as "feasible" if they can be accomplished using relatively low bandwidth and a reasonable amount of time. But, as network speeds are constantly growing, it would be not a problem for an attacker having access to powerful enough uplink to search the entire 32-bit ISN space in several hours, assuming a local LAN connection to the victim host and assuming the network doesn't crash, although an attack could be throttled to compensate. While it seems obvious that OS vendors should do their very best to make such attacks as difficult as possible, it obviously isn't so. In most cases, we are observing ISN generators that are vulnerable to immediate guess, or can be attacked using average modem / DSL connection. This is even in server systems that are supposed to have strong ISNs.

It is worth reminding that risks caused by predictable ISNs were known for at least 15 years, giving more than enough time to build strong and unpredictable implementations. In all PRNG applications where risks were not that emphasized - DNS resolver implementations, authentication cookies and session-id generators, situation is even worse.

Of course, when it comes to TCP/IP ISNs, the good news is that a lot of this is dependent upon certain conditions. For an attack to work, you need the following:

- An open TCP port to get an initial ISN, or a sniffed ISN. If there are no open ports reachable by the

attacker, then the ISN would have to be guessed. Knowing a current ISN would certainly help tailor an attack to increase the odds of its success. This is still very possible, but the level of intelligence required to make the attack in the first place (such as sniffing and probing) would probably point to an alternate attack method that would be far more likely to succeed.

- Routing that will allow forged packets to reach the target. Proper firewall and router rules can prevent forged packets from reaching the target (network interface subnet filtering, expected TTL measuring, ISN storm detection). This does not necessarily mean spoofing is impossible, and, in some cases, might lead to serious DoS conditions, but would make TCP/IP blind spoofing attacks harder to perform.

- A service listening on the target that can be properly exploited via a blind spoof (or, alternatively, existing and known client-server stream that would be vulnerable to spoofing). Except denial of service and potentially exploiting trust relationships, many attacks against listening services often require more than just guessing the ISN. They require the knowledge of several potentially unpredictable factors. If the target doesn't really have an exploitable service or client software running, or some factors required to perform the attack are unknown to the attacker, then his options (ISN spoofing or otherwise) can be extremely limited.

It is worth mentioning that TCP/IP sequence numbers analysis has some other interesting aspects besides finding vulnerable implementations. Every operating system generates a different attractor shape. While metrics used in traditional OS fingerprinting can be easily fooled (usually by altering publicly available kernel parameters via sysctl), a TCP sequence number generator cannot be modified that easily. At the same time, every system seems to have a separate generating function. So, this technique may be modified for reliable remote OS detection, as well.

6. References

TCP/IP networking: http://msdn.microsoft.com/library/backgrnd/html/tcpipintro.htm

TCP/IP spoofing, ISN weakness: http://www.sans.org/infosecFAQ/threats/intro_spoofing.htm

```
Harris, B. and Hunt, R., "TCP/IP security threats and attack methods",
Computer Communications, vol. 22, no. 10, 25 June 1999, pp.
885-897.

Guha, B. and Mukherjee, B., "Network Security via Reverse Engineering
of TCP Code: Vulnerability Analysis and Proposed Solutions",
IEEE Network, vol. 11, no. 4, July/August 1997, pp. 40-48.
```

Phase-space reconstruction:
http://www.mpipks-dresden.mpg.de/~tisean/TISEAN_2.1/docs/chaospaper/node6.html

```
Hegger, R., Kantz, H., and Schreiber, T., "Practical implementation of
nonlinear time series methods: The TISEAN package", Chaos, vol.
9, no. 2, June 1999, pp. 413-435.

Schreiber, T. and Schmitz, A., "Surrogate time series", Physica D,
vol. 142, no. 3-4, 15 August 2000, pp. 346-382.
```

Chaos Theory and Fractal Phenomena: http://www.duke.edu/~mjd/chaos/chaos.html

```
Benoit B. Mandelbrot, "The Fractal Geometry of Nature",
W. H. Freeman and Company, NY, 1977-2000
```


Tool sources: vseq.tgz

Used data samples: data.tgz (approx. 15 MB)

7. Credits

Some of data sets (Solaris, AIX, HPUX) contributed by skyper <skyper@segfault.net>

Windows 95/98 data contributed by noah williamsson <noah@hd.se

Cisco and BSDI data contributed by (anonymous).

Solaris tcp_strong_iss and other data contributed by Elias Levy <aleph1@securityfocus.com>

Router data sets contributed by Piotr Zurawski <szur@ix.renet.pl>

Some of HPUX samples contributed by Solar Designer <solar@openwall.com>

Thanks to Theo de Raadt and all other people who reviewed this publication and came up with suggestions, corrections or comments.

Special thanks to Matt Power, Dave Mann, Mark Loveless, Scott Blake and other RAZOR Team members for their extensive work on this document.

Appendix A: Phase Space Images of Known Generating Functions

Analysis of the attractors associated with PRNGs can be used in another way to create effective spoofing sets. Although this metod does not lead to a universal, automated attack method, it can be easily used to guess and implement the underlying PRNG function.

If we can accurately guess the nature of the generating function that an OS is using for its ISN generator, we can easily build spoofing sets using the same, or at least a very similar, generating function.

We noted that the behavior of attractors is that similar functions will often admit similar attractors. Thus, the attractors of well known PRNGs can be used as a sort of "visual finger print" for the underlying algorithm. If we compare the attractor of a particular OS to a catalog of attractors corresponding to well known PRNGs, we may be able to make reasonable guesses at the type of the PNRG being utilized by the OS.

In practice, the spoof set generation algorithm discussed in section 2.2 is powerful enough that this approach may not need to be employed. Never the less, it is relevant to the general question of reverse engineering so we include a discussion of it for the sake of completeness.

Let's begin by considering some patterns produced by specific algorithms that are frequently used in TCP/IP ISN generators:

```
...2179423068, 2635919226, 2992875947, 3492493181, 2448506773, 2252539982,
2067122333, 1029730040, 335648555, 427819739...
```

32-bit randomness - the perfect ISN generator (smaller, 31-bit cloud would be more suitable, because of RFC requirements). There is no dependency, no regions of attraction, and no path. Data fills every possible space with white noise.

...1946815845, 1931502952, 2004117103, 543519329, 1713402729, 543516018, 544370534, 543976545, 544437353, 757935405...

Statistical correlation - Some regions of phase space are more occupied. In terms of attractor reconstruction, we would say some values are "attracted" by these regions. If a considerably large amount of points can be found near any relatively small region (point, curve, plane, etc.), you can in most cases successfully delimit your search for new values to this region, and you do not have to search the entire available function space. Patterns are different for every "imperfect" algorithm, but if present, they indicate problems.

...4294116998, 3730256972, 4294036130, 3205319395, 4293855531 3568222244, 4294120161, 2206082684, 4294031940...

Partial randomness - Such patterns (planes, empty boxes, etc) are common whenever randomness is mixed with predictable data. This dramatically reduces search space - randomness is still present, but random data is located in easily predictable locations that can be derived from previous results.

...135242143, 135542151, 135642164, 136342181, 135842182, 135542195, 129542200, 135142207, 135542223, 135542257...

Small changes - This pattern is generated for potentially random, but relatively small increments. Such PRNGs are pretty good, except they do not fill the entire available space, and high bits are not as random as they should be. As a result, the attacker has significantly fewer possibilities to examine.

...1699922618, 1699942619, 1699962618, 1699982618, 1700003328, 1700022622, 1700042618, 1700063111, 1700082618...

Trivial dependency (time dependency simulation) - Pictures of this kind usually contain regular groups of points on three surfaces and one very saturated center point where these surfaces are connected. This is a clear sign of a really bad TCP/IP sequence generator.

As in the case of spoofing set reconstruction, this method is a powerful tool for finding flaws and problems with RNGs, but cannot be used to prove a specific implementation is flawless.

---

19 March - 21 April, 2001