Phase space analysis of session cookies

Thomas Sundmark Dinesh Theerthagiri
Email: {thosu588, dinth712}@student.liu.se
Supervisor: David Byers, {davby@ida.liu.se}
Project Report for Information Security Course
Linköpings universitetet, Sweden

Abstract

This report deals with the topic of phase space analysis. It contains information about the experiments we have performed in the area and the conclusions that can be drawn from the outcome of the aforementioned experiments. The main conclusion of these experiments is that the implementation of the PRNGs of Joomla! v1.5.2 running on top of both Ubuntu v7.10 and Windows XP SP2 exhibits weak attractor behavior when analyzed with the method of phase space analysis using delayed coordinates. This means that these systems are well protected against any attacks involving prediction of the upcoming value of a session cookie.

1. Introduction

The purpose of this report is to present the background of phase space analysis as well as to describe the experiments that we have performed in the area and the conclusions that can be drawn from their results.

Phase space analysis is a topic which is very vital in the context of network security as it exposes a weakness in some of the Pseudo Random Number Generators (PRNG:s) used in operating systems as well as in other applications such as DNS-queries and web servers.

The report starts out by presenting the background of phase space analysis and then moves on to explain in detail the experiments we have performed and what the results of these experiments were. Thereafter follows a discussion of the results of the experiments, before the report is finally summarized with a brief conclusion.

2. Background

Phase space analysis using the technique of delayed coordinates is performed by first collecting a significant amount of data. For example, in the case of session IDs in HTTP traffic a large number of generated cookie session ID values are collected.

The next step is to calculate the differences between these sample values, and then use these differences as coordinates in a y-dimensional space.

For example if s[t] is the collection of n sample values and the number of dimensions is three, the relevant x-, y- and z-coordinates are calculated through the following formulas [1, 2]:

$x[t] = s[t] - s[t-1]$
$y[t] = s[t-1] - s[t-2]$
$z[t] = s[t-2] - s[t-3]$

The set of resulting coordinates, also known as the PRNGs function attractor, are then plotted and examined. If the PRNG is weak a clear pattern of points will be displayed, which means that knowledge of three subsequent values of s[n] gives you a high probability of accurately calculating the next number in the sequence by using the aforementioned function attractor.

This is done by constructing what is known as a Spoofing Set. This approach is based on an observation about attractors presented in the report written by Michal Zalewski [1]:

"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."

These Spoofing Sets are usually generated to contain approximately 5,000 points, while the function attractors are generated using up to 50,000 points. This means that one has to gather 50,000 values in advance, possibly spread out over a period of time as to not raise any suspicions. Then at the time of the attack one only needs to gather three values which are then used to generate all the values in the Spoofing Set.

This set of values is generated by first forming the line L through the following equation [1]:

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

This line is then compared to the attractor and if the PRNG is weak the next value in the sequence (seq[t]) will be located somewhere in or near the intersection of these.

However, there is a possibility that the line will not intersect the attractor in any point, which happens when seq[t-3], seq[t-2] and seq[t-1] were not part of the 50,000 values used for constructing the attractor.

This means that in addition to selecting any intersecting points for our Spoofing Set we also want to select points in the attractor that are located "close" to the generated line.

This is performed by selecting all points in the attractor that are located within a radius R1 of the line and then using the values of their x-coordinates to generate values for the Spoofing Set through the following equation: s[t] = s[t-1] + x[t].

As a last step to make sure we get close to 5,000 points in the Spoofing Set without selecting too high of a value for R1 is to take advantage of the fact that the shape of strong attractors (corresponding to weak PRNGs) tend to fill up and become more dense as more points are plotted, implying that the value of x[t] should be relatively close to the x-value of the points already in our Spoofing Set.

Thus we also add the values x+1, x-1… x+R2, x-R2 to our Spoofing Set for each value x already present in the set, where R2 is a value selected to make sure we end up with approximately 5,000 values.

In conclusion, if the PRNG is weak then one of these ~5,000 values in the Spoofing Set is very likely to correspond to the next number generated after the three collected values.

3. Experiments

This section consists of an explanation of the different experiments that were performed in the project.

3.1 Ubuntu v7.10

Our first experiment consisted of trying to use phase space analysis to evaluate the possibility of predicting the upcoming values of the session-cookies generated by Joomla! v1.5.2 [3] when running on the 2.2.8 version of the Apache HTTP server [4] installed on the latest stable version (7.10) of Ubuntu [5].

The tools used for this experiment were the slightly modified 'gathercookies.sh' script [2] for gathering the cookie values (see Appendix A for further details) and the standard text formatting tool Notepad for removing the irrelevant non-changing parts of the responses.

The remaining values were then fed as input to the 'hexstring2bigintstring.pl' script [2] to convert them from a hexadecimal representation to a more easily handled integer format before being fed to the 'seq.pl' script [2] which calculated the delayed coordinates corresponding to the collected cookies.

The coordinates were then saved in a .dat file which was in turn plotted using gnuplot v4.2.3 [6].

The output from the hexstring2bigintstring.pl script was also fed into the calprob [1] script which performed the actual analysis of the number generator by taking as input the file containing the integer representation of the cookies, the desired value of R1, the size of the spoofing set as well as the amount of attempts that should be made.

3.2 Windows XP SP2

In our second experiment we performed the analysis on the same version of Joomla! [3] and an Apache HTTP server [4] but we replaced the operating system with Windows XP SP2 [7].

With the exception of the cookie collection process we performed the analysis in the same way as above, by first converting the cookie values to an integer format, then fed those integers into the seq.pl and calprob scripts before finally plotting the coordinates using gnuplot [6].

The reason the cookie collection process differed was because we were unable to get an automated script working for collecting the cookie values and thus had to perform the collection manually.

3.3 Custom PRNG

Our third and final experiment involved writing our own implementation of a PRNG and analyzing it using the same tools as in the previous experiments.

The purpose of this experiment was to show an example of a weak PRNG due to the fact that, as our results below will show, both the initial experiments yielded results which relate to strong PRNG implementations.

4. Results

This section contains the results of the different experiments performed throughout the project.

4.1 Ubuntu v7.10

Plotting the coordinates generated by the first experiment yielded the image shown in Figure 1 below.

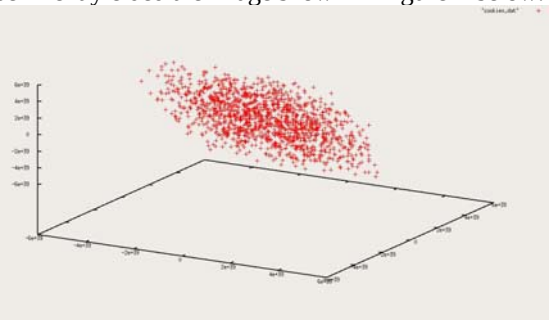

Figure 1: First experiment

The analysis using the calprob script yielded the following results regardless of the specified value of R1:

Failed attempts: 0/10 (0%)
Average R2: 4998
Average N: 11
Probability: 0%

Table 1: Results

The failed attempts refer to how many of the attempted predictions failed to give any output for the given value of R1, while the probability is based on how close those outputs came to the actual value given the size of R2.

The average N value corresponds to the average amount of generated guesses for the given R1 value.

4.2 Windows XP SP2

For the second experiment gnuplot generated the image shown in Figure 2 below.

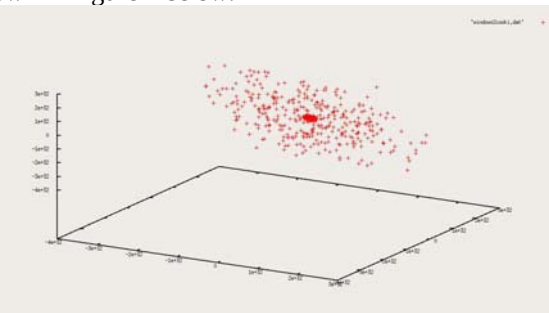

Figure 2: Second experiment

Analysing the results with the help of calprob gave the following results for all attempted values of R1:

Failed attempts: 0/10 (0%)
Average R2: 4998
Average N: 11
Probability: 0%

Table 2: Results

See the note below Table 1 in the previous subsection for an explanation of the different values.

4.3 Custom PRNG

Plotting the coordinates yielded by this experiment resulted in the following figure:

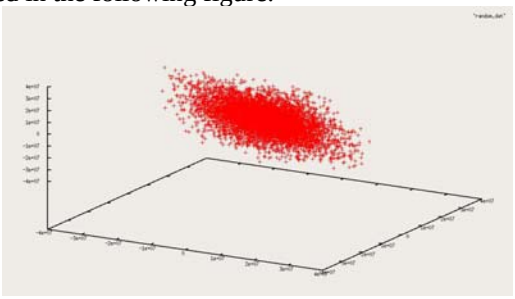

Figure 3: Third experiment

The analysis using calprob yielded different results depending on the choice of R1.

Failed attempts: 423/500 (84%)
Average R2: 2352
Average N: 10
Probability: 15%

Table 3: Results - R1 = 1000

Failed attempts: 209/500 (41%)
Average R2: 1555
Average N: 21
Probability: 58%

Table 4: Results - R1 = 10000

Failed attempts: 20/500 (4%)
Average R2: 468
Average N: 131
Probability: 96%

Table 5: Results - R1 = 100000

An explanation of the values can be found below Table 1 in the results subsection for the first experiment.

5. Discussion

The results of the first experiment show that the combination of Joomla! v1.5.2 [3], Apache v2.2.8 [4] and Ubuntu v7.10 [5] has a good implementation of its PRNG for generating cookies.

The second experiment using Windows XP SP2 [7] instead of Ubuntu v7.10 [5] still shows signs of a correct implementation of a PRNG for generating the cookies even though its cloud is slightly smaller, with a scale of 10^32 compared to the scale of 10^39 for the cloud from the first experiment.

The difference in the scales of the two operating systems is quite significant, but since the PRNG used by Joomla! [3] running on Windows XP SP2 [7] still has very good randomness the difference doesn't matter much. It would still take a very large amount of attempts to be able to accurately predict the value of an upcoming session cookie.

The third experiment on the other hand shows that a simple PRNG such as the one we wrote using the C++ function rand() and very simple modulo arithmetic, can easily be broken given a high enough value of R1. It should also be noted that even with R1 being as low as 1000 there is still a possibility to correctly predict the upcoming number in the sequence.

6. Conclusions

In this report we have given a basic introduction to phase space analysis using delayed coordinates, provided details of three experiments that we performed using this technique and presented the results of the experiments before rounding off the report with a short discussion of the results.

The performed experiments showed that both Ubuntu v7.10 [5] and Windows XP Service Pack 2 [7] have a strong PRNG implementation when it comes to generating session cookies for Joomla! v1.5.2 [3] running on an Apache v.2.2.8 [4] web server.

However, the Ubuntu v7.10 [5] implementation appears slightly stronger due to its attractor shape being significantly larger with a scale of 10^39 compared to 10^32 for Windows XP SP 2 [7].

In conclusion one can see that the PRNG implementations of both systems investigated in the experiments causes the probability of a successful attack against them using phase space analysis with delayed coordinates to be close to zero.

References

[1] M. Zalewski, "Strange Attractors and TCP/IP Sequence Number Analysis", http://lcamtuf.coredump.cx/oldtcp/tcpseq.html, 2001. (Accessed 2008-02-22)

[2] Florian Walther, "How to analyse a Session ID", http://www.xs4all.nl/%7Escusi/SessionID-release/www/index.html, 2006. (Accessed 2008-02-22)

[3] Joomla!, http://www.joomla.org/ (Accessed 2008-04-14)

[4] Apache HTTP Server Project, http://httpd.apache.org/ (Accessed 2008-04-14)

[5] Ubuntu Home Page, http://www.ubuntu.com/ (Accessed 2008-04-14)

[6] Gnuplot homepage, http://www.gnuplot.info/ (Accessed 2008-04-14)

[7] Windows XP: Home Page, http://www.microsoft.com/windows/products/windowsxp/default.mspx (Accessed 2008-04-23)

Appendix A: Tools

Gathercookies.sh

The ‘gathercookies.sh’ script available for download at [2] was modified by replacing the ‘grep ASPSESSIONID’ line with ‘grep Set-Cookie:’ to produce the following script:

```
#!/bin/sh
# gather.sh
while [ 1 ]
     do
     echo -e "GET / HTTP/1.0\n\n" |  \
     nc -vv $1 80 | \
     grep Set-Cookie:
     done
```

Hexstring2bigintstring.pl

The ‘hexstring2bigintstring.pl’ script available for download at [2] was used to convert the cookies from hexadecimal numbers to an integer representation.

```
use Math::BigInt lib => 'GMP';

while(<>) {
          chomp;
          $inp = $_;
          $inp =~ s/ //g;
          $sum = Math::BigInt->new();
          $add = Math::BigInt->new();
          $mul = Math::BigInt->new();
          $exp = Math::BigInt->new('1');
          $counter = 0;
          if (length($inp) % 2 == 1) {
            $inp = "0" . $inp;
          }
          for ($i = length($inp) - 2; $i >= 0; $i
-= 2) {
            $conv = substr($inp, $i, 2);
            $char = pack('H2', $conv);
            $ord = ord($char);
            $add->bzero();
            $add->badd($ord);
            $mul->bzero();
            $mul->badd($add);
            if ($counter > 0) {
              $exp->bmul('256');
            }
           $counter += 1;
            $mul->bmul($exp);
            $sum->badd($mul);
#           print "conv: " . $conv . ", add: " .
$add . ", exp: " . $exp . ", sum: " . $sum .
"\n";
          }
          print $sum . "\n";
}
```

Seq.pl

The ‘seq.pl’ script available for download at [2] was used to calculate the delayed coordinates for the converted cookies values given as output from the format conversion script.

```
use Math::BigInt;

@seq = ();
@x = @y = @z = ();
while(<>) {
        chomp($val = $_);
        if (length($val) > 0) {
          push(@seq, $val);
        }
}
for ($i = 3; $i < $#seq; $i++) {
        $o = Math::BigInt->new($seq[$i]);
        $m1 = Math::BigInt->new($seq[$i-1]);
        $m2 = Math::BigInt->new($seq[$i-2]);
        $m3 = Math::BigInt->new($seq[$i-3]);
        push(@x, scalar $o->bsub($m1));
        push(@y, scalar $m1->bsub($m2));
        push(@z, scalar $m2->bsub($m3));
}


for ($i = 0; $i < $#seq; $i++) {
        if (defined $x[$i]) {
          print $x[$i] . " " . $y[$i] . " " .
$z[$i] . "\n";
        }
}
```

Calprob

The ‘calprob’ script provided by M. Zalewski [1] was modified slightly to become compatible with the more recent versions of Bash, yielding the following script:

```
#!/bin/sh

if [ "$4" = "" ]; then
  echo "Usage: $0 datafile.gz r1 fixed_set_size
do_tests"
  exit 1
fi

gzip -cd $1 >.tmp3
RADIUS=$2
SIZ=$3
MAX=$4

SUCC=0
SIZE=`grep -c . .tmp3`
FAILED=0
R2SUM=0
NSUM=0
ERSUM=0
NMSUM=0
```

```
for i in `seq 1 $MAX`; do
  SEQ1=`tail -$((5+$MAX-$i)) .tmp3 | head -1`
  SEQ2=`tail -$((4+$MAX-$i)) .tmp3 | head -1`
  SEQ3=`tail -$((3+$MAX-$i)) .tmp3 | head -1`
  SEQ4=`tail -$((2+$MAX-$i)) .tmp3 | head -1`
  echo "$i: Trying $SEQ1 $SEQ2 $SEQ3 (-> $SEQ4)
- r=$RADIUS"
  cat .tmp3 | head -$(($SIZE/2)) | ./guess3d
$SEQ1 $SEQ2 $SEQ3 $RADIUS | grep '^> '| awk -F'>
' '{print $2}' | sort -n >.points
  QQ=`cat .points`
  if [ "$QQ" = "" ]; then
    echo "  - radius R1 too small? No answers!"
    FAILED=$(($FAILED+1))
    continue
  fi
  CNT=`cat .points|wc -c|awk '{print $1}'`
  USER2=`./rsort $SIZ <.points 2>/dev/null`
  echo "  + guess3d gave $CNT answers, rsort
suggests R2 of $USER2..."
  GOT=0
  AVV=0
  for i in $QQ; do
    AVV=$(($AVV+1))
    ERROR=`echo "if ($i-$SEQ4>0) { $i-$SEQ4 }
else { $SEQ4-$i }" | bc -q`
    if [ "$ERROR" -le "$USER2" ]; then
      GOT=1
      break
    fi
  done
  R2SUM=$(($R2SUM+$USER2))
  NSUM=$(($NSUM+$CNT))
  if [ "$GOT" = "1" ]; then
    SUCC=$(($SUCC+1))
    NMSUM=$(($NMSUM+$AVV))
    #ERSUM=$(($ERSUM+$ERROR))
    echo "  -> SUCCESSFUL (difference $ERROR)."
#    exit 0
  else
    echo "  -> Bad guess..."
  fi
done

echo
echo "Data file:         $1"
echo "Failed attempts:   $FAILED/$MAX
($(($FAILED*100/$MAX))%)"
echo "Average R2:        $(($R2SUM/($MAX-
$FAILED)))"
echo "Average N:         $(($NSUM/($MAX-
$FAILED)))"
if [ "$SUCC" = "0" ]; then
  echo "No stats available"
else
  echo "Average error:     $(($ERSUM/$SUCC))"
  echo "Average correct N: $(($NMSUM/$SUCC))"
fi
echo "Probability:
$(($SUCC*100/$MAX)).`printf '%04d'
$(($SUCC*100000/$MAX % 1000))`%"

rm -f .tmp3 .points .tmp1 .tmp2
```

guess3d.c

The source code for the ‘guess3d’ program used by the ‘calprob’ script is the following:

```
#include <vgagl.h>
#include <stdio.h>
#include <stdlib.h>

struct ientry {
  unsigned int x,y,z;
};

struct ientry itable[100000];
unsigned int it1[100000];
int cnt;
unsigned int p1,p2,p3;
unsigned int nz,ny;
int gsm,i;
int fr;
unsigned int val;
int radius=0;
unsigned int known[100000];

main(int argc,char* argv[]) {
  unsigned int tmp;
  FILE* x;
  if (argc<4) {
    printf("Usage: guess3d seq[t-3] seq[t-2]
seq[t-1] [radius] <datafile\n");
    printf("Will guess seq[t].\n");
    exit(1);
  }
  p1=strtoul(argv[1],0,10);
  p2=strtoul(argv[2],0,10);
  p3=strtoul(argv[3],0,10);
  if (argc==5) {
    radius=atoi(argv[4]);
  } else radius=0;
//  printf("%u %u %u\n",p1,p2,p3);
  printf("Reading input data...\n");
  while (scanf("%u",&tmp)==1) {
    it1[cnt++]=tmp;
    if (cnt>=100000) break;
  }
  for (i=3;i<cnt;i++) {
    itable[i].x=it1[i]-it1[i-1];
    itable[i].y=it1[i-1]-it1[i-2];
    itable[i].z=it1[i-2]-it1[i-3];
//    printf(" %u %u
%u\n",itable[i].x,itable[i].y,itable[i].z);
  }
  ny=p3-p2;
  nz=p2-p1;
  printf("Reconstructed: x={guess set} y=%u z=%u
(radius=%d)\n",ny,nz,radius);
  printf("Choosing candidates (%d
entries)\n",cnt);
  x=fopen(".tmp","w+");
  for (i=3;i<cnt;i++) {
   int j;
   unsigned int distance=fabs(itable[i].z-
nz)+fabs(itable[i].y-ny);
   if (distance<=radius) {
      fprintf(x,"%u
%u\n",distance,p3+itable[i].x);
    }
  }
  fclose(x);
  printf("Sorting candidates...\n");
  system("cat .tmp | sort -n | uniq -c | sort -n
-r >.tmp2");
```

```
  x=fopen(".tmp2","r");
  while (fscanf(x,"%d %u %u",&fr,&fr,&val)==3)
gsm++;
  fclose(x);
  if ((gsm>cnt/20) || (gsm<1)) {
    printf("Attractor unsuitable [%d].\n",gsm);
    exit(1);
  }
  printf("Guess set (%d elements, from most
probable to least):\n",gsm);
  x=fopen(".tmp2","r");
  cnt=0;
  while (fscanf(x,"%d %u %u",&fr,&fr,&val)==3) {
    for (i=0;i<cnt;i++) if (known[i]==val)
break;
    if (i==cnt) {
      known[i]=val;
      printf("> %u\n",val);
      cnt++;
    }
  }
//  unlink(".tmp"); unlink(".tmp2");
}
```

rsort.c

The source code for the ‘rsort’ program used by the ‘calprob’ script is the following:

```
#include <stdio.h>
#include <unistd.h>

#define MAXPOINTS 10000

unsigned int start,stop,oldstart,oldstop;
int count;
unsigned int points[MAXPOINTS];
int i,j;
unsigned int over;
char buf[1000];
int r;
int dumpanddie;
unsigned int tmp;
main(int argc,char* argv[]) {
  if (argc-2) { printf("Usage: guess3d set_size
<data\n"); exit(0); }
  r=atoi(argv[1]);
  while (scanf("%u",&tmp)==1) {
    // Input has to be sorted (sort -n)!
    points[count]=tmp;
    count++;
  }
  for (i=0;i<r;i++) {
loopa:
    over=0; oldstart=0; oldstop=0;
    for (j=0;j<count;j++) {
      start=points[j]-i;
      if (i>points[j]) start=0;
      stop=points[j]+i;
      if (stop<start) stop=0xffffffff;
//      printf("Current range: %u-%u (old %u-
%u).\n",start,stop,oldstart,oldstop);
      if (start<=oldstop) {
          // Joint ranges.
//         printf(" + Joining ranges!\n");
          oldstop=stop;
          continue;
      }
      if (dumpanddie) fprintf(stderr,"Range %u-
%u.\n",oldstart,oldstop);
      over+=(oldstop-oldstart+1);
      oldstart=start; oldstop=stop;
    }
    over+=(oldstop-oldstart+1);
    if (dumpanddie) { fprintf(stderr,"Range %u-
%u.\n",oldstart,oldstop); exit(0); }
//    printf("For i=%d set size is
%u.\n",i,over);
    if (over>=r) { printf("%d\n",i);
dumpanddie=1; goto loopa; }
  }
}
```