دوازده روز کریسمس

شعر دوازده روز کریسمس تحت عنوان نغمه سرایی کریمس انگلیسی ها نیز شناخته می شود کـه افـراد در ایـن ایام به یکدیگر یادبودها و هدایایی را می دهند. این دوازده روز، از شب کریسمس آغاز شده و در انتهای شب دوازدهم بـا شروع Epiphany خاتمه می یابد که پایان مراسم سنتی کریسمس را نیز خبـر میدهـد. شـعر دوازده روز کریـسمس یـک آهنگ تراکمی است، به این معنی که هر قطعه شعر بر اساس قطعات قبلی ساخته می شود. دوازده قطعه شعر وجود دارند که هر کدام اعطای یک هدیه را به یک عشق حقیقی بیان می دارند. اولین قطعه به صورت زیر است:

On the first day of Christmas, my true love sent to me
A partridge in a pear tree.

دومین قطعه نیز در زیر قرار دارد:

On the second day of Christmas, my true love sent to me
Two turtle doves
and a partridge in a pear tree.

و به همین ترتیب ... . قطعه آخر نیز به صورت زیر می باشد:

On the twelfth day of Christmas, my true love sent to me
Twelve drummers drumming,
eleven pipers piping,
ten lords a-leaping,
nine ladies dancing,
eight maids a-milking,
seven swans a-swimming,
six geese a-laying,
five gold rings;
four calling birds,
three French hens,
two turtle doves
and a partridge in a pear tree.

این نغمه آنقدر محبوب است که حتی یک شاهکار عظیم برنامه نویسی نیز از آن نشات گرفته است: یک کد رمزی شده.

کد رمزشده (یا obfuscated – از تضاد با واژه encode یـا encrypt جلـوگیری کنیـد)، کـد منبعـی اسـت کـه خواندن و فهمیدن آن بسیار دشوار است. بعضی زبان های برنامه نویـسی طبعـا بـرای اعمـال فرآینـدهای Obfuscation خاصیت و ویژگی های خاصی دارند که C، C++ و Perl از معمول تـرین آنهـا اسـت. ماکروهـای پـیش پردازنـده[1] اغلـب جهت تولید کدهای ناخوانا استفاده می شوند که این کار را با پوشاندن ساختار (syntax) استاندارد زبان از گونه اصلی کـد انجام می دهند. واژه *کد پوشش دار* یا *Shrouded Code* اغلب برای توصیف اینطـور کـدها بکـار مـی رود. برنامـه هـایی تحت عنوان Obfuscator وجود دارد که روی کد منبع عملیاتی را انجام می دهند و در نهایت کار را بـرای انجـام عملیـات reversing دشوار می سازند.

[1] Macro pre-processor

مبهم سازی به روش تولید مجدد

یک کد بعضی مواقع عمدا جهت اهداف باز آفرینی مبهم می شود. چندین مفاهیم برنامه نویسی وجود دارد که تاثیرات بسزایی در عملیات مبهم سازی کد دارند. انواع گوناگونی از روش های جالب مبهم سازی وجود دارد، از جمله تعویض ساده کلمات کلیدی، استفاده/عدم استفاده از Whitespace ها و ... .

در زبان پرل، جمله ای تحت عنوان "Just Another Perl Hacker" معروفیت خاصی پیدا کرد، چرا که اولین جمله ای بود که در زبان پرل مبهم سازی می شد و معمولا آنرا در امضای برنامه نویس های حرفه ای پرل کم و بیش می یابید (البته به غیر از ایران)!! کد مذکور را می توانید با جستجو در گوگل بیابید و روی آن کار کنید.

هدف ما آنالیز برنامه دوازده روز کریسمس است که در زبان برنامه نویسی C نوشته شده است. در زیر کد منبع مربوط به این برنامه را می بینید:

```
#include <stdio.h>
main(t,_,a)char *a;{return!0<t?t<3?main(-79,-13,a+main(-87,1-_,
main(-86,0,a+1)+a)):1,t<_?main(t+1,_,a):3,main(-94,-27+t,a)&&t==2?_<13?
main(2,_+1,"%s %d %d\n"):9:16:t<0?t<-72?main(_,t,
"@n'+,#'/*{}w+/w#cdnr/+,{}r/*de}+,/*{*+,/w{%+,/w#q#n+,/#{l,+,/n{n+,/+#n+,/#\
;#q#n+,/+k#;*+,/'r :'d*'3,}{w+K w'K:'+}e#';dq#'l \
q#'+d'K#!/+k#;q#'r}eKK#}w'r}eKK{nl]'/#;#q#n'){)#}w'){){nl]'/+#n';d}rw' i;# \
){nl]!/n{n#'; r{#w'r nc{nl]'/#{l,+'K {rw' iK{;[{nl]'/w#q#n'wk nw' \
iwk{KK{nl]!/w{%'l##w#' i; :{nl]'/*{q#'ld;r'}{nlwb!/*de}'c \
;;{nl'-{}rw]'/+,}##'*}#nc,',#nw]'/+kd'+e}+;#'rdq#w! nr'/ ') }+}{rl#'{n' ')# \
}'+}##(!!/")
:t<-50?_==*a?putchar(31[a]):main(-65,_,a+1):main((*a=='/')+t,_,a+1)
  :0<t?main(2,2,"%s"):*a=='/'||main(0,main(-61,*a,
"!ek;dc i@bK'(q)-[w]*%n+r3#l,{}:\nuwloca-O;m .vpbks,fxntdCeghiry"),a+1);}
```

با اولین انگاه این کد غیر مفهوم و پیچیده به نظر می آید، اما در عمل یک برنامه قانونی و بدون نقص در زبان C است که به هنگام اجرا و کامپایل ۱۲ قطعه مربوط به شعر معروف "دوازده روز کریسمس" را تولید خواهد کرد. این کد عملا حاوی تمام رشته های مورد نیاز برای این شعر است که به صورت رمزی در کد جاسازی شده است. سپس کد بواسطه این دوازده روز تکرار شده و هر بار کامل تر می گردد.

ما می دانیم که برنامه چه کاری را انجام می دهد. سوالی که وجود دارد این است که این برنامه چگونه چاپ این شعر را انجام می دهد. البته ما بسادگی می توانیم برنامه ای طراحی کنیم که شعر دوازده روز کریسمس را چاپ کند، اما این کار دقیقا بمانند این می ماند که شما برای خودتان هدیه ای بخرید، درحالیکه قبلا کسی برای شما هدیه ای خریده است!!

در عملیات مهندسی معکوس روی برنامه، ما مصمم هستیم که اثرات محاسباتی برنامه را تا جای ممکن همان گونه که هست حفظ کنیم. در صورت امکان، ما برنامه را به حالت برنامه اولیه می نویسیم و اینطور نیست که هر تکه مختلف از کد را مجزا برای خود طراحی کنیم بدون اینکه بدانیم در برنامه اصلی چه مواردی در حال رویداد است.

حالت رسمی دوازده روز کریسمس

قبل از پرداختن به عملیات مهندسی معکوس، داشتن یک مدل بسیار مهم است؛ چرا که با استفاده از آن فرآیند مهندسی معکوس را هموارتر و بدیهی تر می سازیم (No Plan, No work)! شعر دوازده روز کریسمس تماما درباره هدایا می باشد، لذا ما شعر را با تشخیص مقادیری مطالعه می کنیم که از ساختار طبیعی شعر ناشی می شوند. با یک نگاه سریع به این نتیجه می رسیم که شعر O(N*N) بار برای خواندن و به O(N*N) قدر فضا برای نوشتن نیاز دارد که در آن N تعداد هدایا می باشد (در این مثال، 12). اکنون تعداد دقیق دفعاتی که هدایا در شعر ذکر شده اند را بیابیم که کار ساده ای است: هدیه ای که برچسب t را داشته باشد، 13-t بار در شعر ذکر شده است. به این صورت:

بر چسب هدیه	تعداد دفعاتی که هدیه در طول ۱۲ روز ارائه می شود
T	13-T

بعنوان مثال، هدیه "۵ حلقه طلایی" (five gold rings) به تعداد 13-5 یعنی ۸ بار رخ داده است. با جمع تمام هدایا می توانیم کل هدیه هایی که در طول این ۱۲ روز داده شده است را بدست آوریم:

1+2+…+11+12 = 13 * 6 = 78

یعنی در طول این ۱۲ روز کلا ۷۸ هدیه داده شده است. با کم کردن تعداد هدیه های non-partridge (منظور هدیه روز اول که تعداد آن در کل شعر ۱۲ است) از کل هدیه ها عدد ۶۶ نتیجه می شود که این عدد تعداد هدایای تکراری را بازگو می کند.

آنالیز شعر این نکته را نشان می دهد که هر قطعه بیشترین رشته های ابتدایی و انتهایی را با تغییر پذیری در روز و لیست هدیه ها در یک روز معین دارد. در زیر الگویی را ارائه می کنیم که این مسئله را نشان می دهد:

1). On the <*x*> day of Christmas my true love gave to me
2). *<list of gifts from the <x> day down to the second day>*
3). and a partridge in a pear tree.

در این الگو، x شماره روز است: خط یک در تمامی قطعه ها تکرار می شود. خط دوم لیست هدیه های موجود بین آنروز و روز دوم را چاپ می کند و خط آخر نیز هدیه روز اول را می نویسد. اکنون مقداری با کارکرد برنامه آشنا شدیم. ولی یک احتمال منفی را نیز در شعر بررسی می کنیم که اولین قطعه می باشد:

On the first day of Christmas my true love gave to me
a partridge in a pear tree.

همان طور که می بینید خروجی چاپ شده دقیقا مطابق الگویی که در بالا پیشنهاد کردیم نیست، چرا که در الگوی بالا کلمه "and" در ابتدای جمله بکار رفته است، در حالیکه در اینجا حرف "a" می باشد. به این صورت می توانیم با استفاده از الگوی فوق، تعداد دقیق رشته های منحصر بفردی را که جهت چاپ صحیح شعر، درون برنامه جاسازی شده است را بدست آوریم. این تعداد از قرار زیر هستند:

- سه رشته برای ساختار اصلی ("On The"، "day of Christmas…" و "and a partridge…")
- دوازده رشته برای ترتیب ها و وصف ها

- یازده رشته برای هدیه های روزهای دوم تا دوازدهم (12-2+1=11 – مراقب به اشتباه محاسباتی off-by-one باشید تا حاصل ۱۰ بدست نیاید).

به این ترتیب احتمالا برنامه دارای 3+12+11 = 26 عدد رشته یکتا، و حدودا 3*12 + 12 + 66 = 114 عدد رشته میباشد. می توانیم برنامه ای بنویسیم که تعداد کاراکترهای خروجی برنامه را نیز بدست آورد. با انجام این کار، تعداد کاراکترهای خروجی برنامه را ۲۳۵۸ عدد یافتیم. لذا شروع واقعی مطالعه خود روی کد، اعداد مهمی که خط مشی عملیات ما را هدایت می کنند به صورت زیر هستند:

- ۱۲ روز کریسمس (این عدد را ۱۱ نیز در نظر می گیریم تا موارد "off-by-one" را نیز بدست آوریم).
- ۲۶ رشته یکتا
- ۶۶ رخداد مربوط به هدایایی non-partridge (منظور هدیه اول)
- ۱۱۴ رشته چاپ شده
- ۲۳۵۸ کاراکتر چاپ شده

طراحی یک برنامه خوانا

برای درک یک برنامه، خواندن آن بسیار مفید خواهد بود. برنامه مورد مطالعه ما تقریبا غیرقابل خواندن است، حتی برای افرادی که آشنایی و تسلط بسیار بالایی با زبان C دارند. هدف اصلی در وهله اول، انتقال این کد به یک Compiler C و سپس پرانتز گذاری آن برای افزایش قابلیت خوانایی کد می باشد. نتیجه این عمل را در زیر می بینید:

```
#include <stdio.h>
main(t,_,a)
char *a;
{
  return
    ((!0) < t )
      ? ((t < 3
          ? main(-79,-13,a+main(-87,1-_,main(-86,0,a+1)+a))
          : 1),

         (t < _
          ? main(t+1,_,a)
          : 3),

         (main(-94,-27+t,a)
          && (t==2
              ? ( _ < 13
                  ? main(2,_+1,"%s %d %d\n")
                  : 9)
              : 16)))

        : (t < 0
           ? (t < -72
              ?
              main(_,t,
                   "@n'+,#'/*{}w+/w#cdnr/+,{}r/*de}+,/*{*+,/w{%+,/w#q#n+,/#{l+
,/n{n+,/+#n+,/#\
;#q#n+,/+k#;*+,/'r :'d*'3,}{w+K w'K:'+}e#';dq#'l \
q#'+d'K#!/+k#;q#'r}eKK#}w'r}eKK{nl]'/#;#q#n'){)#}w'){){nl]'/+#n';d}rw' i;# \
){nl]!/n{n#'; r{#w'r nc{nl]'/#{l,+'K {rw' iK{;[{nl]'/w#q#n'wk nw' \
```

```
iwk{KK{nl]!/w{%'l##w#' i; :{nl]'/*{q#'ld;r'}{nlwb!/*de}'c \
;;{nl'-{}rw]'/+,}##'*}#nc,',#nw]'/+kd'+e}+;#'rdq#w! nr'/ ') }+}{rl#'{n' ')# \
}'+}##(!!/")

              : (t < -50
                 ? (_ == *a
                    ? putchar(31[a])
                    : main(-65,_,a+1))
                 : main((*a=='/')+t,_,a+1)))
           : (0 < t
              ? main (2,2,"%s" )
              : * a=='/'
              || main(0,main(-61,*a,
                              "!ek;dc i@bK'(q)-[w]*%n+r3#l,{}:\nuwloca-
O;m .vpbks,fxntdCeghiry"),a+1)
              )
           );
}
```

اما هنوز هم چندان دلچسب نیست، چرا که نویسنده کد بجای استفاده از جملات if-then-else و بلوک جملات، از عبارات شرطی (e1 ? e2 : e3) و لیست عبارات (e1, e2, e3) استفاده کرده است. لذا، ما برنامه را دقیقا با همان کارایی اما بدون استفاده از عبارات شرطی یا لیست عبارات مجددا می نویسیم تا برنامه نسبتا خوانای زیر را بدست آوریم:

```
#include <stdio.h>
main(t,_,a)
     char *a;
{
  if ((!0) < t) {
    if (t < 3)
      main(-79,-13,a+main(-87,1-_,main(-86,0,a+1)+a));

    if (t < _ )
      main(t+1,_,a);

    if (main(-94,-27+t,a)) {
      if (t==2 ) {
        if ( _ < 13 ) {
          return main(2,_+1,"%s %d %d\n");
        } else {
          return 9;
        }
      } else
        return 16;
    } else
      return 0;

  } else if (t < 0) {
    if (t < -72) {
      return main(_,t,
                  "@n'+,#'/*{}w+/w#cdnr/+,{}r/*de}+,/*{*+,/w{%+,/w#q#n+,/#{l+,
/n{n+,/+#n+,/#\
;#q#n+,/+k#;*+,/'r :'d*'3,}{w+K w'K:'+}e#';dq#'l \
q#'+d'K#!/+k#;q#'r}eKK#}w'r}eKK{nl]'/#;#q#n'){)#}w'){){nl]'/+#n';d}rw' i;# \
){nl]!/n{n#'; r{#w'r nc{nl]'/#{l,+'K {rw' iK{;[{nl]'/w#q#n'wk nw' \
iwk{KK{nl]!/w{%'l##w#' i; :{nl]'/*{q#'ld;r'}{nlwb!/*de}'c \
;;{nl'-{}rw]'/+,}##'*}#nc,',#nw]'/+kd'+e}+;#'rdq#w! nr'/ ') }+}{rl#'{n' ')# \
}'+}##(!!/");
    } else if (t < -50 ) {
      if (_ == *a) {
        return putchar(31[a]);
      } else {
        return main(-65,_,a+1);
      }
    } else {
      return main((*a=='/')+t,_,a+1);
```

```
    }

  } else if (0 < t) {
    return main (2,2,"%s" );

  } else {
    if (*a=='/')
      return 1;
    else return
      main(0,main(-61,*a,
                  "!ek;dc i@bK'(q)-[w]*%n+r3#l,{}:\nuwloca-
O;m .vpbks,fxntdCeghiry"),a+1);
  }
}
```

بعلاوه، ما چندین قطعه از کد (که به آنها کد مرده می گوییم) را نیز حذف کردیم (چون در عملیات آنالیز جایگاهی نداشتند). برای مثال، در ترجمه لیست عبارت زیر مقادیر ثابت ۱ و ۳ که در کد بصورت درشت نما وجود دارند، کدهای مرده محسوب می شوند، چرا که این مقادیر توسط هیچ یک از عبارات دیگر مصرف نشده و مورد استفاده قرار نمی گیرند:

```
((t < 3
 ? main(-79,-13,a+main(-87,1-_,main(-86,0,a+1)+a))
 : 1),

(t < _
 ? main(t+1,_,a)
 : 3),

(main(-94,-27+t,a)
        && (t==2
            ? ( _ < 13
                ? main(2,_+1,"%s %d %d\n")
                : 9)
            : 16)))
```

لذا عبارات اول و دوم در لیست عبارات به صورت زیر ترجمه می شوند:

```
if (t < 3)
  main(-79,-13,a+main(-87,1-_,main(-86,0,a+1)+a));

if (t < _ )
  main(t+1,_,a);
```

ساختار برنامه

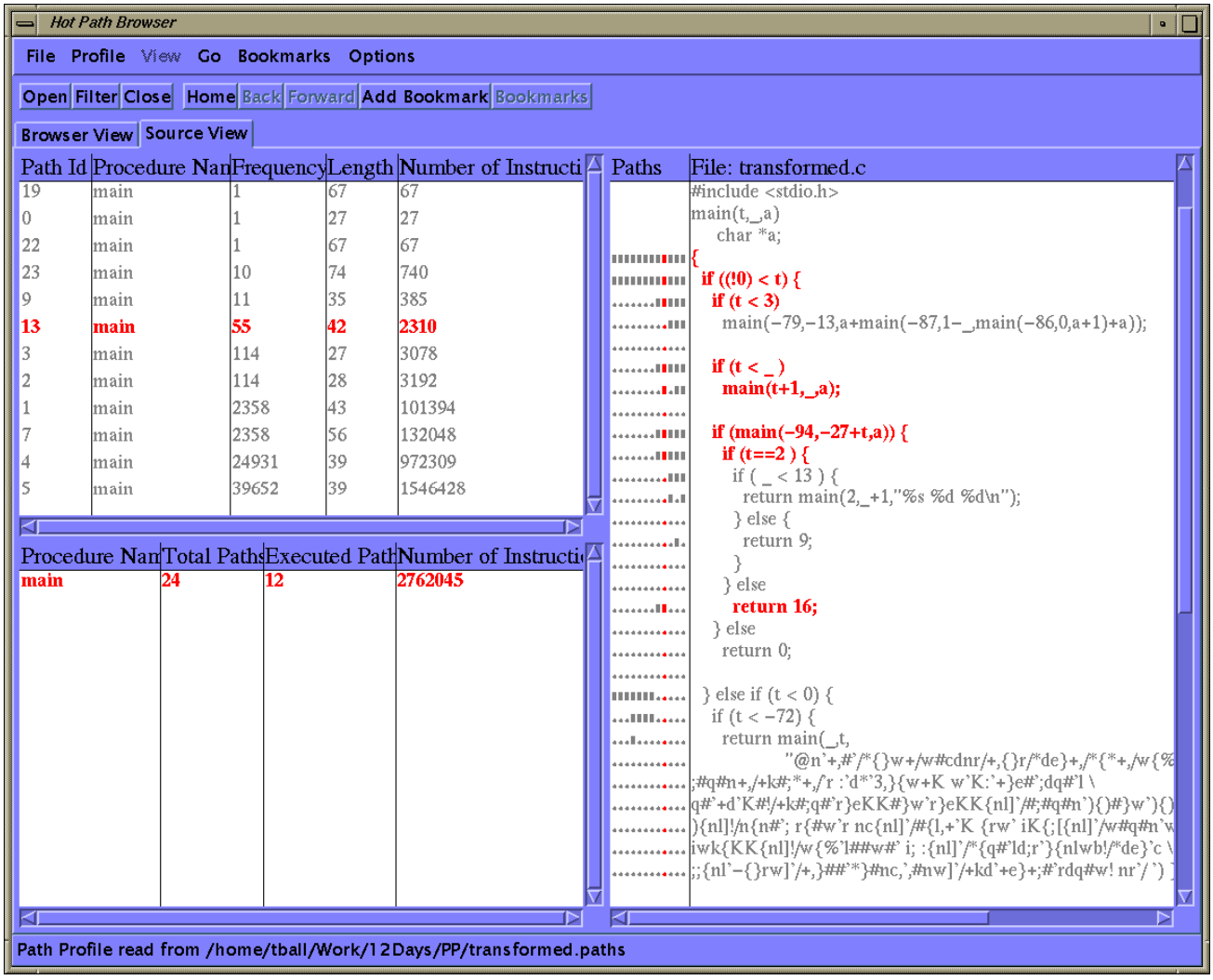
همان طور که می بینیم، برنامه خوانای ما دارای یک تابع main حاوی سه آرگومان است و خودش را مکررا فراخوانی می کند. تابع main اهداف خود را منحصرا مبنی بر مقادیر منتقل شده به خود بدست می آورد. اولین آرگومان منتقل شده به main، تعداد آرگومان های خط فرمان است (که شامل نام خود برنامه نیز می شود). بنابراین، هنگامی که ما برنامه را بدون آرگومان اجرا می کنیم، مقدار پارامتر t برابر با ۱ خواهد بود. هیچ حلقه صریحی وجود ندارد؛ بلکه این حلقه بواسطه عملیات بازگشتی (recursion) روی تابع main بوجود می آید. انتخاب پایه های تابع احتمالا بواسطه پارامتر t انجام میشود. برنامه حاوی دو رشته بزرگ است که ظاهرا متن شعر را رمزگذاری می کنند.

انجام عملیات Path Profiling روی برنامه خوانا

همان طور که به یاد دارید، الگوی ما استفاده از مقادیر مختلف بدست آمده از خروجی برنامه جهت تست و درک برنامه بود. لذا ما از یک ابزار profiling جهت ذخیره شمارش اجرایی برنامه خوانای خود استفاده می کنیم. طرز فکر ما این است که این شمارش های اجرایی ما را در فهم این نکته یاری می کنند که کدام قسمت های برنامه مسئول کدام قسمت های شعر هستند. برای مثال، یک عنصر برنامه با شمارش اجرایی معادل با ۱۱ یا ۱۲ ممکن است وجود یک هویت را در کنترل تعداد قطعه های شعر نشان دهد، در حالیکه یک عنصر با شمارش اجرایی برابر با ۲۳۵۸ احتمالا در عملیات چاپ کاراکترها بکار رفته است.

در حالت کلی، یک ابزار Profiling، خرده کدهایی را به یک برنامه اضافه می کند که تعداد دفعات هویت ها را در یک اجرای معین از برنامه می شمرد (از قبیل جملات، عبارات و غیره و ذالک). بخصوص، ما ابزار Hot Path Browser (HPB) از Ball،Larus و Rosa را بر روی برنامه تغییر شکل یافته (برنامه واضح) اعمال کردیم. این ابزار به برنامه ها دستور می دهد که پروفایل های Ball/Larus Path را ضبط کرده و نمایش دهند. یک پروفایل Ball/Larus Path تعداد دفعات اجرای هر مسیر غیرحلقه ای (و درون رویه ای) را می شمارد.

تصویر زیر اعمال ابزار HPB را روی برنامه خوانا نشان می دهد:



پنجره سمت چپ-بالا موجود در این ابزار، آمار و محاسبات مربوط به هر مسیر اجرا شده را نمایش می دهد. بر حسب تصادف می بینیم که دوازده مسیر (از بین ۲۳ مسیر ممکن) اجرا شده است!! مسیرها به صورت صعودی بر حسب تکرار لیست شده اند. مسیر با شناسه سیزده انتخاب شده است (خط قرمز). این مسیر در قسمت نمایش کدمنبع در سمت راست highlight شده است. چگونگی گردآوری (کلاستر کردن) مسیرها را بر حسب تکرار به خاطر بسپارید. با یک آزمایش دستی روی کد، ما امضای محاسباتی هر کلاستر از مسیرها را تشخیص داده ایم.

- مسیر صفر (یک بار اجرا شده است) فرآیند بازگشت را با فراخوانی main(2,2,…) شروع می کند.
- مسیرهای ۱۹، ۲۲ و ۲۳ کنترل چاپ دوازده قطعه را بر عهده دارند. یعنی مسیر ۱۹ اولین قطعه، مسیر ۲۳ ده قطعه وسطی و مسیر ۲۲ آخرین قطعه را نمایش می دهند. مجموع تکرار این مسیرها برابر با ۱۲ است. همان طور که در تصویر مربوط به مسیرها مشاهده میشود، هر یک از این مسیرها یک مجموعه متفاوت از فراخوانی های بازگشتی را به تابع main برعهده دارند. این مسیرها ما را در جهت پی بردن به این نکته کمک کردند که کدام فراخوانی ها مسئول اولین خط هر قطعه هستند (فراخوانی سه باره به main و تکرار بیرونی ترین حلقه (main(2,_+1,…)).
- مسیرهای ۹ و ۱۳ کنترل چاپ هدیه های non-partridge را درون یک قطعه بر عهده دارند. به خاطر داشته باشید که جمع تکرارهای این دو مسیر برابر با ۶۶ می شود.
- مسیرهای ۲ و ۳ مسئول چاپ یک رشته هستند. هر مسیر ۱۱۴ بار تکرار دارد. این عدد، همان تعداد دقیق رشته هایی است که ما در مدل خود آنرا پیش بینی کردیم.
- مسیرهای ۱ و ۷ کاراکترهای موجود در یک رشته را چاپ می کنند. هر مسیر ۲۳۵۸ بار اجرا می شود. شاید این سوال پیش بیاید که چرا این دو مسیر وجود دارند؟ بعدا به این مسئله نیز اشاره خواهیم کرد.
- راجع به مسیرهای ۴ و ۵ با تکرارهای زیاد و عجیب و غریبی برابر با ۲۴۹۳۱ و ۳۹۶۵۲ چه فکری می کنید؟! آزمایش کد نشان می دهد که مسیر ۴ مسئولیت پرش از روی n زیر-رشته در رشته بزرگ را بر عهده دارد. هر زیر-رشته با کاراکتر '/' خاتمه یافته است. هر بار که یک زیر-رشته چاپ می شود، یک جستجوی خطی در سرتاسر متن رشته جهت یافتن رشته انجام می شود؛ از اینرو است که مسیر ۴ تکرار فوق العاده بالایی دارد.
- مسیر ۵ به صوت خطی رشته ای را که ترجمه کاراکتر را رمزگذاری می کند پویش کرده تا کاراکتری که با کاراکتر فعلی مطابق است بیابد و در نهایت آنرا چاپ نماید. این عمل برای چاپ کردن هر کاراکتر انجام می شود. مسیر ۵ قسمتی از عملیات محاسباتی است که مسیرهای ۱ و ۷ را نیز درگیر کار می کند و بایستی با آنها کلاستر یا دسته بندی شود.

این آنالیز شش کلاستر اصلی از مسیرها را نشان داد.

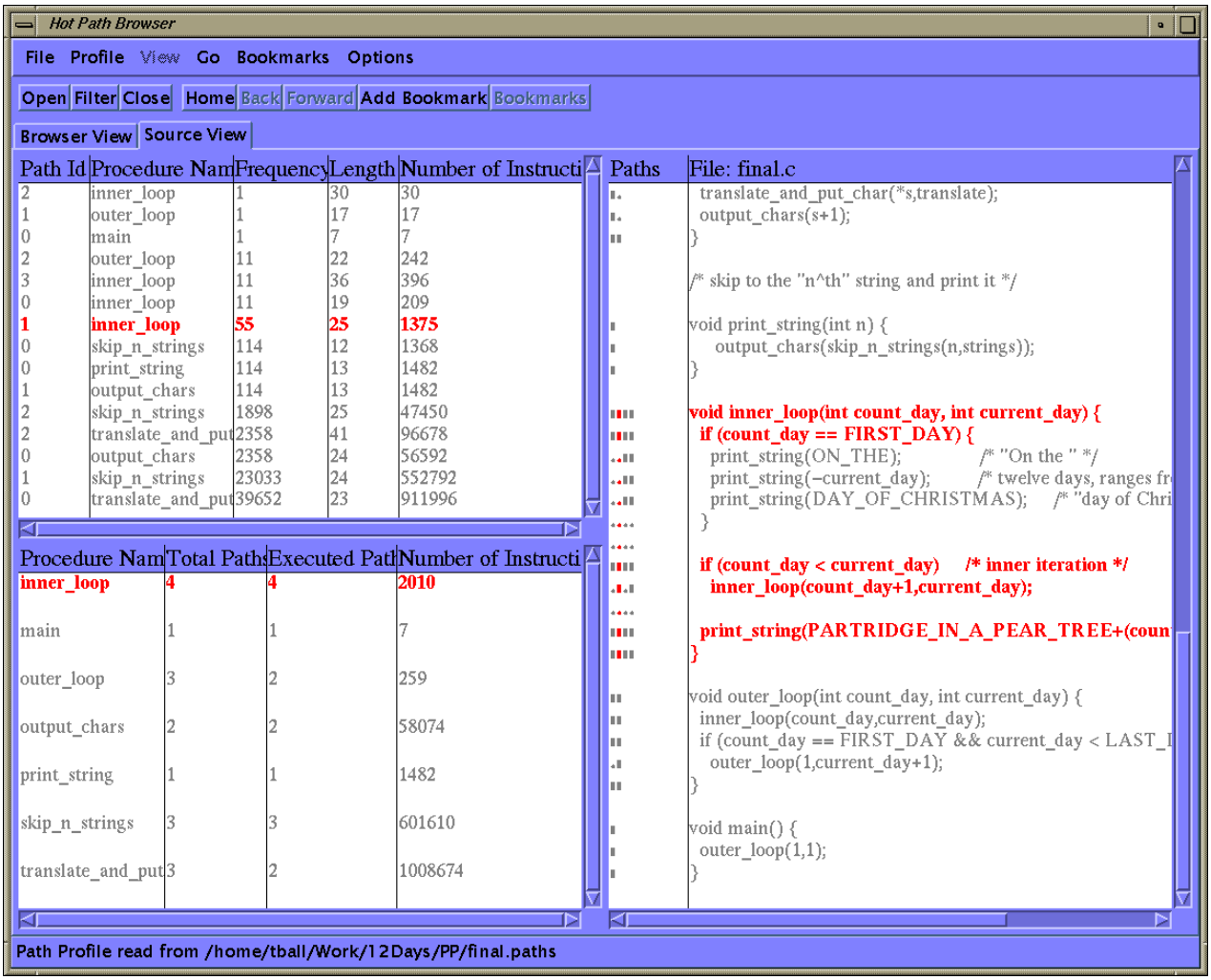
عملیات مهندسی معکوس روی برنامه

با استناد به اطلاعاتی که از پروفایل های مسیر و آزمایش دستی برنامه بدست آمد، ما روی برنامه عملیات مهندسی معکوس را انجام می دهیم. ما سعی می کنیم ساختار بازگشتی برنامه را سالم و دست نخورده نگاه داریم، که این کار را فقط برای استفاده از توابع مختلف جهت نمایش وظایف گوناگون از برنامه اولیه انجام دادیم همان طور که با استفاده از کلاسترکردن مسیرها ضبط و ذخیره شد. ما مقادیر دو رشته متن مربوطه را به هیچ وجه تغییر ندادیم (لیست رشته ها و

نگاشت ترجمه برای رمزگذاری کاراکترها). برنامه اصلی از مقدار ۲ برای نمایش اولین روز کریسمس استفاده می کرد. ما این مقدار به ۱ شیفت کردیم تا مطابق شعر باشد. هفت تابع در برنامه جدید وجود دارند که عینا متناظر با شش کلاستر مسیری هستند که در بالا تشخیص داده شدند:

- `main` (path 0)
- `outer_loop` (paths 19, 22 and 23)
- `inner_loop` (paths 9 and 13)
- `print_string` (paths 2 and 3)
- `output_chars` (paths 1 and 7) and `translate_and_put_char` (path 5)
- `skip_n_strings` (path 4)

برنامه جدید خروجی دقیقا مشابه با برنامه اولیه را دارد و همین طور تمام اشکالات اجرایی. برای اینکه نشان دهیم ذات و جوهره اصلی برنامه را ذخیره کرده ایم، ما پروفایل مسیر را روی برنامه جدید اعمال می کنیم تا بررسی کنیم آیا تکرار مسیر در برنامه جدید با برنامه اولیه یکسان است یا خیر. تصویر زیر که مربوط به برنامه جدید می باشد نشان می دهد که مسیر آن تقریبا با برنامه اصلی یکسان است، اگرچه ۱۵ مسیر از کل ۱۷ مسیر اجرا شده اند:



برای اغلب مسیرها، این یک تناظر یک به یک با یک مسیر در کد اصلی می باشد. برای مثال، مسیر انتخاب شده در تصویر مربوط به برنامه جدید (مسیر ۱ از رویه inner_loop) تکراری برابر با ۵۵ بار دارد. این نتایج مشابه نتایج محاسباتی روی مسیر ۱۳ در تصویر مربوط به برنامه اولیه می باشند.

مرور

برنامه "دوازده روز کریسمس" از سه حقه ساده جهت مخفی نگه داشتن اهداف خود استفاده می کند:

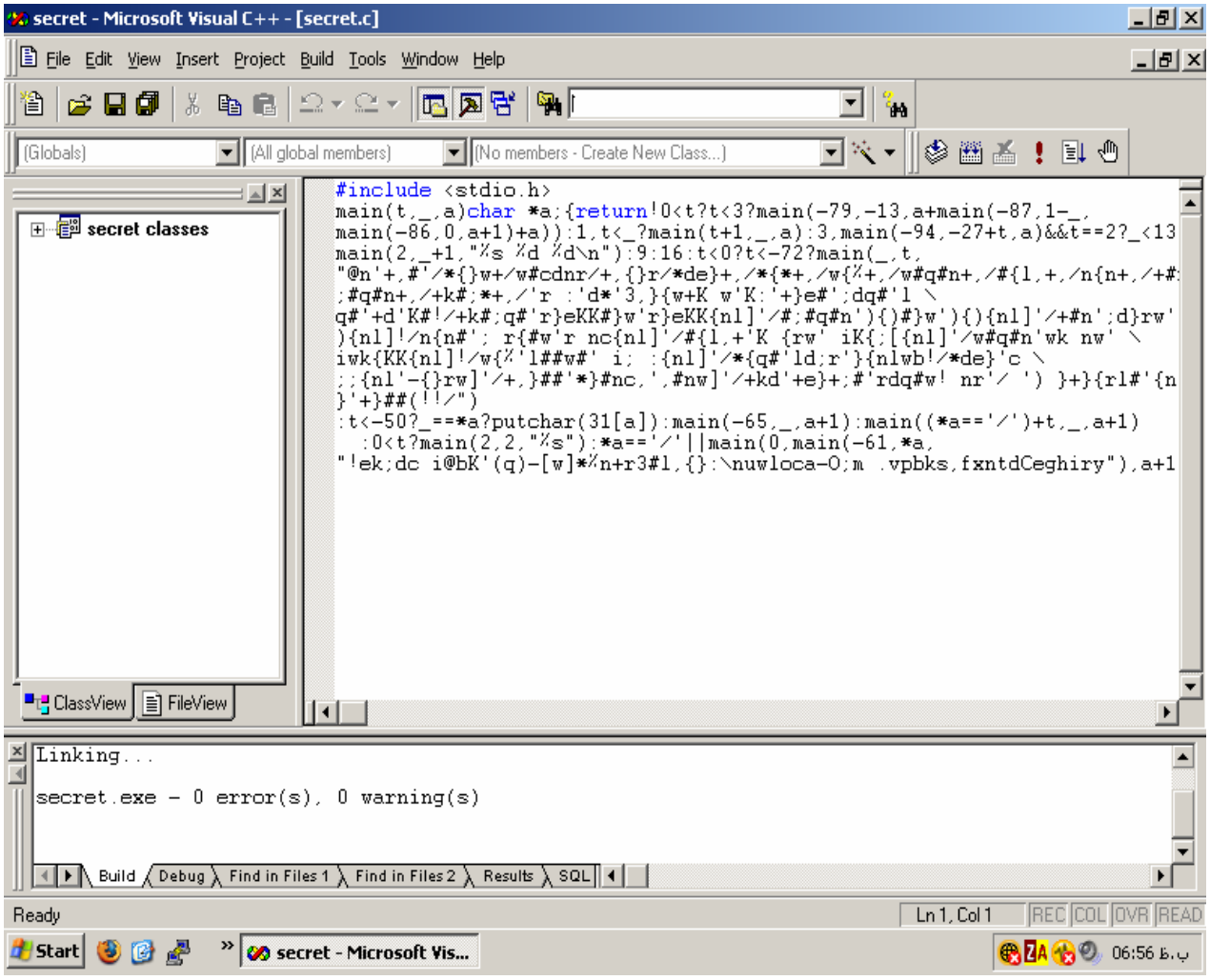
- **حذف تمام** whitespace **ها**
- **استفاده از عبارات و لیست های شرطی به جای استفاده از جملات آشنا و دوستانه تر** if-then-else **و بلوک ها.**
- **رمزگذاری ساده از رشته های شعر**

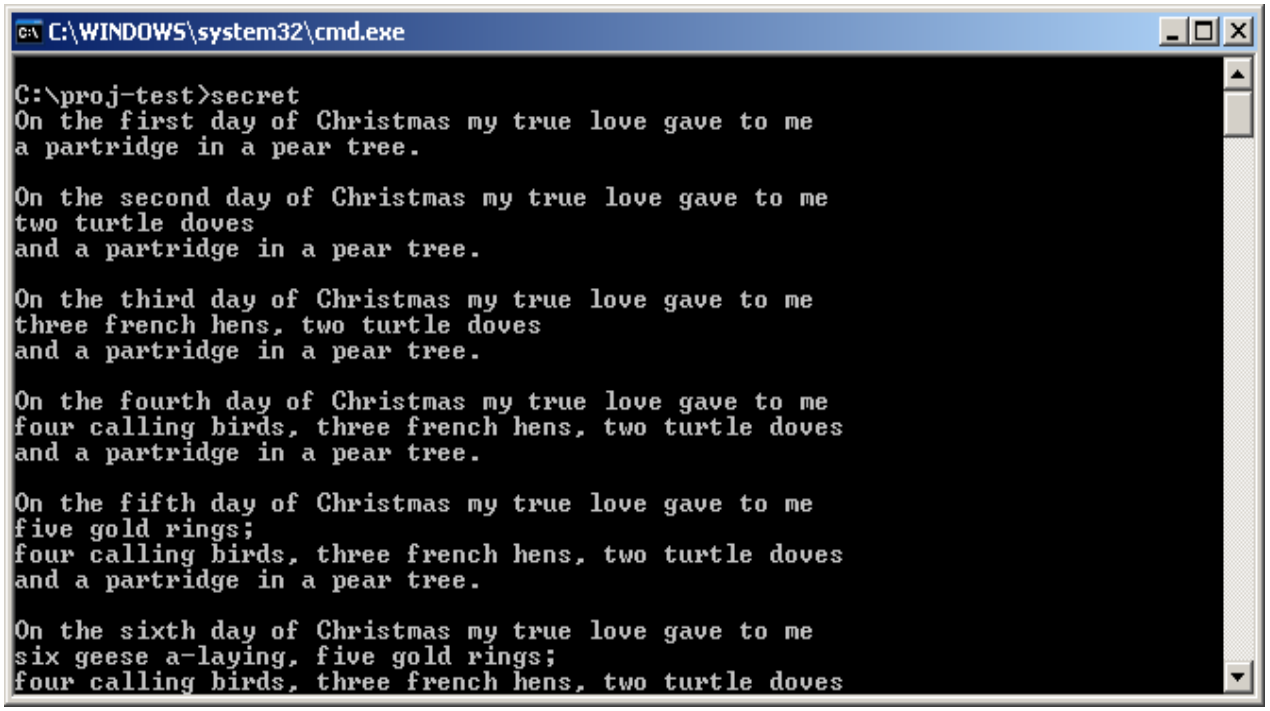
به هر حال، موردی که درک برنامه را بسیار دشوار می سازد، رمزگذاری چندین "تابع" درون تابع واحد main **می باشد. یک قضیه شناخته شده در امنیت کامپیوتر وجود دارد که میگوید: هر برنامه را که بتوان درون یک برنامه معنادار و معادل آن از یک رویه تغییر شکل داد، حاوی یک جمله** switch **در یک حلقه** while **می باشد! تغییر شکل شامل بازنویسی مجدد برنامه نیز می شود بطوریکه هر** case **از** switch**، یک عملگر مقدماتی از برنامه اولیه است که با یک تخصیص به یک متغیر شمارشی برنامه دنبال است که به این صورت دستور بعدی برای اجرا را به جمله** switch **متصل می کند. حلقه** while **تعیین می کند که چه هنگام شمارشگر برنامه به آخرین عملگر در برنامه رسیده است. نتیجه این برهان علمی این است که هر برنامه را که بتوان به صورت برنامه ای بازنویسی کرد که حاوی یک تابع بازگشتی واحد باشد، تنها حاوی جملات شرطی خواهد بود؛ که این مورد در رابطه به کد مورد مطالعه ما در این مقاله سازگار بود. اولین پارامتر تابع** main**، نقش شمارشگر برنامه را برای رنج مشخصی از مقادیر بازی می کند. همچنین تعیین می کند که کدام رشته چاپ شود. ما از تکنولوژی** Path Profiling **جهت جدا کردن توابع کاملی (منظور غیر ناقص) استفاده کردیم که این تابع واحد پیاده سازی می کند. البته این کار با استناد به این موضوع انجام شد که شعر چیزی جز یک پروفایل از هدایای دریافتی نخواهد بود! در زیر برنامه ای که عملیات مهندسی معکوس را روی آن انجام دادیم را مشاهده می کنید:**

```
#include <stdio.h>

static char *strings =

"@n'+,#'/*{}w+/w#cdnr/+,{}r/*de}+,/*{*+,/w{%+,/w#q#n+,/#{l+,/n{n+,/+#n+,/#\
;#q#n+,/+k#;*+,/'r :'d*'3,}{w+K w'K:'+}e#';dq#'l \
q#'+d'K#!/+k#;q#'r}eKK#}w'r}eKK{nl]'/#;#q#n'){)#}w'){){nl]'/+#n';d}rw' i;# \
){nl]!/n{n#'; r{#w'r nc{nl]'/#{l,+'K {rw' iK{;[{nl]'/w#q#n'wk nw' \
iwk{KK{nl]!/w{%'l##w#' i; :{nl]'/*{q#'ld;r'}{nlwb!/*de}'c \
;;{nl'-{}rw]'/+,}##'*}#nc,',#nw]'/+kd'+e}+;#'rdq#w! nr'/ ') }+}{rl#'{n' ')# \
}'+}##(!!/";

static char *translate =

"!ek;dc i@bK'(q)-[w]*%n+r3#l,{}:\nuwloca-O;m .vpbks,fxntdCeghiry";

#define FIRST_DAY  1
#define LAST_DAY  12

enum {
   ON_THE = 0,
   FIRST = -1,
   TWELFTH = -12,
   DAY_OF_CHRISTMAS = -13,
   TWELVE_DRUMMERS_DRUMMING = -14,
   ELEVEN_PIPERS_PIPING = -15,
   TWO_TURTLE_DOVES_AND_A = -24,
   PARTRIDGE_IN_A_PEAR_TREE = -25
};
```

```
char* skip_n_strings(int n,char *s) {
  if (n == 0)
    return s;

  if (*s=='/')
    return skip_n_strings(n+1,s+1);
  else
    return skip_n_strings(n,s+1);
}

void translate_and_put_char(char c, char *trans) {
  if (c == *trans)
    putchar(trans[31]);
  else
    translate_and_put_char(c,trans+1);
}

void output_chars(char *s) {
  if (*s == '/')
    return;
  translate_and_put_char(*s,translate);
  output_chars(s+1);
}

void print_string(int n) {
     output_chars(skip_n_strings(n,strings));
}

void inner_loop(int count_day, int current_day) {
  if (count_day == FIRST_DAY) {
    print_string(ON_THE);                /* "On the " */
    print_string(-current_day);          /* twelve days, ranges from -1 to -12
*/
    print_string(DAY_OF_CHRISTMAS);      /* "day of Christmas ..." */
  }

  if (count_day < current_day)      /* inner iteration */
    inner_loop(count_day+1,current_day);

  print_string(PARTRIDGE_IN_A_PEAR_TREE+(count_day-1));   /* print the gift */
}

void outer_loop(int count_day, int current_day) {
  inner_loop(count_day,current_day);
  if (count_day == FIRST_DAY && current_day < LAST_DAY)  /* outer iteration */
    outer_loop(1,current_day+1);
}

void main() {
  outer_loop(1,1);
}
```

تصاویر کامپایل و اجرای اولیه برنامه





نویسنده: سعید بیکی (cephexin@secumania.net)

© Secumania Security & Vulnerability Research Lab
www.secumania.net