



РЕШЕНИЯ ЗАДАЧ XII МЕЖРЕГИОНАЛЬНОЙ ОЛИМПИАДЫ ШКОЛЬНИКОВ ПО ИНФОРМАТИКЕ И КОМПЬЮТЕРНОЙ БЕЗОПАСНОСТИ



2017 год

Задача 1. Хеш-значения

Политика безопасности ОС не позволяет задавать для учетных записей пользователей пароли, совпадающие с их именами. Для этого перед добавлением нового пользователя в базу вызывается функция `CheckUser()`. При ее успешном выполнении (возвращаемое значение = 0) в базу добавляется новая запись, содержащая имя учетной записи пользователя и хеш-значение пароля, полученное с помощью функции `Hash()`.

Си	Паскаль
<pre>int CheckUser(char* username, char* password) { for (int i=0; i<strlen(username); i++) { if ((username[i] >= 0x30 && username[i] <= 0x39) (username[i] >= 0x41 && username[i] <= 0x5A) (username[i] >= 0x61 && username[i] <= 0x7A)) continue; else return 1; } for(int i=0; i<strlen(password); i++) { if (password[i] >= 0x23 && password[i] <= 0x7D) continue; else return 1; } if (strcmp(username, password) != 0) return 0; else return 1; }</pre>	<pre>function CheckUser(var username: string; password: string) : integer; var i:integer; begin for i:=1 to Length(username) do begin if(((username[i] >= #48) and (username[i] <= #57)) or ((username[i] >= #65) and (username[i] <= #90)) or ((username[i] >= #97) and (username[i] <= #122))) then continue else begin Result := 1; Exit; end; end; for i:=1 to Length(password) do if((password[i] >= #35) and (password[i] <= #125)) then continue else begin Result := 1; Exit; end; if(UpperCase(username)<> UpperCase(password)) then Result :=0 else Result :=1; end;</pre>
<pre>const char letters[] = "abcdefghijklmnopqrstuvwxyABCDEFGHIJKLMNOPQRSTUVWXYZ01234567890"; char* Hash(char* str) { const int p = 19;</pre>	<pre>function Hash(str:string[255]): string[20]; var Nstr, hsize:integer; var i, p_hash, p_pow, p:word; var buf:string[255]; var str_size, hash_size:integer; var letters:string[100];</pre>

```

int Nstr = 32;
const int Hsize = Nstr / 2;
unsigned short int hash = 0, p_pow = 1;
char *buf = new char[Nstr + 1];
int StrSize = 0;
int HashSize = 0;
char *res = new char[Hsize + 1];

while (str[StrSize] != '\0')
{
    if (HashSize >= Nstr)
        HashSize = Nstr - 1;
    buf[HashSize] = str[StrSize];
    StrSize++;
    HashSize++;
}
for (int i = HashSize; i < Nstr; i++)
{
    buf[i] = 'A' + (i - HashSize);
}
buf[Nstr] = '\0';
for (int i = 0; i < Nstr; i += 2)
{
    hash += (buf[i + 1] - 'A' + 1) * p_pow;
    p_pow *= p;
    hash += (buf[i] - 'A' + 1) * p_pow;
    p_pow *= p;
    res[i / 2] = letters[hash %
strlen(letters)];
}
res[Hsize] = '\0';
return res;
}

```

```

var res:string[32];
begin
    letters:='abcdefghijklmnopqrstuvwxy
ABCDEFGHIJKLMNPOQRSTUVWXYZ01234567890';
    p:=19;
    Nstr:=32;
    Hsize:=Nstr div 2;
    p_hash:=0;
    p_pow:=1;
    str_size:=1;
    hash_size:=1;
    while ( str_size<=length(str) ) do
    begin
        if (hash_size > Nstr) then
        begin
            delete(buf,hash_size,1);
            hash_size:=Nstr;
        end;
        buf:=buf+str[str_size];
        str_size:=str_size+1;
        hash_size:=hash_size+1;
    end;
    for i:=hash_size to Nstr+1 do
        buf:= buf+chr(ord('A')+(i-
hash_size));
    i:=1;
    repeat
        p_hash:=p_hash+(ord(buf[i+1])-
ord('A')+1)*p_pow;
        p_pow:=p_pow*p;
        p_hash:=p_hash+(ord(buf[i])-
ord('A')+1)*p_pow;
        p_pow:=p_pow*p;
        res:=res+letters[p_hash mod
length(letters)+1];

        i:=i+2;
    until i>Nstr;
    Hash:=res;
end;

```

Администратор периодически выполняет проверку базы пользователей и блокирует учетные записи, хеш-значения от имени которых совпадают с хеш-значениями их паролей. Приведите пример имени и пароля для учетной записи, которая удовлетворяет заданной политике безопасности, но будет заблокирована администратором в ходе проверки, и обоснуйте почему.

Решение:

Проанализировав функцию CheckUser(), можно сделать вывод о том, что она действительно не позволяет создавать пользователей, пароль которых совпадает со значением имени его учетной записи. Следовательно, причина возникновения нарушений связана с ошибкой в вычислении хеш-значения пароля.

Действительно, нарушения при добавлении пользователя в базу могли произойти из-за возможности возникновения коллизий при использовании функции Hash() (ситуаций, когда для некоторых различных данных значения хеш-функции совпадают) Рассмотрим одну из причин возникновения коллизии.

Из анализа кода:

```

1. while (str[StrSize] != '\0')
2. {
3.     if (HashSize >= Nstr)
4.         HashSize = Nstr - 1;
5.     buf[HashSize] = str[StrSize];
6.     StrSize++;

```

```
7.  HashSize++;
8. }
```

видно, что в случае если длина пароля превышает 32 символа, то его 32 символ заменяется на последний символ пароля. При этом в случае если длина пароля менее 32 символов, то для вычисления хеш-значения его длина увеличивается до 32 символов:

```
1. for (int i = HashSize; i < Nstr; i++)
2. {
3.     buf[i] = 'A' + (i - HashSize);
4. }
```

В дальнейшем, в вычислении результирующего хеш-значения (переменная *res*) участвуют только первые 32 символа пароля (или его расширенной до 32 символов копии):

```
1. for (int i = 0; i < Nstr; i += 2)
2. {
3.     hash += (buf[i + 1] - 'A' + 1) * p_pow;
4.     p_pow *= p;
5.     hash += (buf[i] - 'A' + 1) * p_pow;
6.     p_pow *= p;
7.     res[i / 2] = letters[hash % strlen(letters)];
8. }
```

Следовательно, если длина пароля будет превышать 32 символа, то начиная с 33 позиции его символы не будут участвовать в формировании хеш-значения, за исключением последнего символа, значение которого присвоится 32 символу. Таким образом, возникновение ситуации, когда хеш-значения от имени учетной записи пользователя и заданного для него пароля совпадают, возможно, например, при одновременном выполнении следующих условий:

- пароль отличается от имени;
- длина имени 32 символа или более;
- длина пароля больше 32-х символов, но заканчивается на 32-й символ;
- первые 32 символа имени и пароля совпадают.

Ответ: Можно указать любую пару Имя-Пароль, удовлетворяющую следующим критериям:

- Пароль отличается от Имени;
- длина Имени 32 символа или более;
- длина Пароля больше 32-х символов, но заканчивается на 32-й символ;
- первые 32 символа Имени и пароля совпадают.

Например:

имя – 123...901...01...012;

пароль – 123...901...01...012xxx2 (xxx – любое количество любых символов).

Задача 2. Секретное сообщение

В исполняемый файл PROG.EXE было внедрено секретное текстовое сообщение. При этом сам файл корректно выполняет все функции. Известно, что для того, чтобы отметить место внедрения информации, нарушитель использовал метку размером 1 байт:

Метка (1 байт)	Сообщение	Метка (1 байт)
-------------------	-----------	-------------------

Какое сообщение было внедрено в файл?

К задаче прилагается: исполняемый файл PROG.EXE.

Решение:

Для однозначного определения внедренного сообщения необходимо в файле найти байт, который встречается в нем ровно 2 раза. Для оптимизации процесса поиска необходимо реализовать программу подсчета встречаемости байтов в файле. Единственный байт, который может выступать в роли метки, имеет значение 0xE1. Данные между метками — сообщение. После извлечения соответствующих байтов их необходимо перевести в символы по ASCII-таблице.

Ответ: леснагоре

Задача 3. Антивирус

Для выявления вредоносного кода некоторым антивирусом применяется только сигнатурный метод анализа, позволяющий выполнять поиск известных сигнатур в файле путем побайтового сравнения. Файл считается вредоносным при наличии в нем участка данных, точно совпадающего с одной из сигнатур. Реализация поиска сигнатур описана в функции CheckFile(), которая возвращает TRUE при отсутствии сигнатур в файле и FALSE в противном случае.

Си	Паскаль
<pre> /* ВХОДНЫЕ ПАРАМЕТРЫ: * FNAME - имя анализируемого файла * SIGNATURE - сигнатура (массив байтов) * SIZE - размер сигнатуры в байтах * * ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ: * TRUE - файл не содержит сигнатуры * FALSE - файл содержит хотя бы одну * сигнатуру */ bool CheckFile(char* fname, char* signature, int size) { FILE *fin=NULL; char *buf=NULL; int read, check_count=0; buf=new char[size]; fin=fopen(fname, "rb"); if (!fin) return false; while(!feof(fin)) { read=fread(buf, 1, size, fin); if(read!=size) break; check_count=0; for (int j=0; j<size; j++) { if (buf[j]==signature[j]) check_count++; else break; } if (check_count==size) return false; } return true; } </pre>	<pre> /* ВХОДНЫЕ ПАРАМЕТРЫ: * FNAME - имя анализируемого файла * SIGNATURE - сигнатура (массив байтов) * SIZE - размер сигнатуры в байтах * * ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ: * TRUE - файл не содержит сигнатуры * FALSE - файл содержит хотя бы одну * сигнатуру */ function CheckFile (fname, signature: string; size: integer): boolean; var fin: file; ch: char; count_read, check_count, i: integer; buf: string; begin assign(fin, fname); reset(fin); while true do begin buf:=''; count_read:=0; for i:=1 to size do begin if (not eof(fin)) then begin read(fin, ch); buf:=buf+ch; count_read:=count_read+1; end; end; if count_read <> size then break; check_count:=0; for i:=1 to size do begin if(buf[i]=signature[i]) then check_count:=check_count+1 else break; end; end; end; end; end; </pre>

```
end;  
if check_count=size then  
begin  
    CheckFile:=false;  
    Exit;  
end;  
end;  
close(fin);  
CheckFile:=true;  
end;
```

База сигнатур содержит две записи:

0x313132

0x3131

Проверка файла осуществляется последовательным вызовом функции `CheckFile()` для каждой сигнатуры из базы.

Какое количество файлов размером 7 байт, состоящих только из цифр от 0 (код 0x30) до 9 (код 0x39) включительно, может содержать хотя бы одну из указанных сигнатур хотя бы один раз, но успешно проходить проверку имеющимся антивирусом? Ответ обоснуйте.

Решение:

Из анализа функции `CheckFile()` видно, что антивирус анализирует файл, начиная с нулевого байта, блоками данных размером по 3 байта, без пересечений. Поэтому, если начало сигнатуры в одном блоке, а конец – в другом, то она не будет обнаружена.

Для оптимизации процесса подсчета количества файлов можно написать программу, которая реализует корректный поиск сигнатур с использованием функции, полученной из `CheckFile()` добавлением строки

```
fseek(fin, -(read - 1), SEEK_CUR);
```

в конец цикла `while`.

Разница между числом файлов, проходящих проверку антивирусами на основе функции `CheckFile()` и ее исправленной версии, – искомое количество файлов.

Ответ: 238.950

Задача 4. Архив

Школьник скачал с некоторого Интернет-ресурса архив PROGS.RAR, который, согласно приведенному на сайте описанию, содержит пакет простых утилит (каждой утилите соответствует ровно один исполняемый файл формата .EXE). После распаковки архива школьник обнаружил, что часть файлов из архива зашифрована методом «двоичного гаммирования», т.е. путем выполнения операции «побитового исключающего ИЛИ» между байтами исходного файла и байтами, полученными циклическим повторением последовательности некоторого ключа.

Зашифрованные файлы не запускаются, а при попытке запуска незашифрованных файлов, некоторые из них блокируются антивирусом из-за наличия в них подозрительной сигнатуры «0x0B0A0C0F».

Помогите школьнику получить из архива максимальное количество программ, которыми он сможет воспользоваться, не отключая антивирус.

К задаче прилагается: архив PROGS.RAR, скаченный школьником с Интернет-ресурса.

Решение:

На первом этапе необходимо выделить множество файлов, которые корректно запускаются: Clockres.exe; plink.exe; pslist.exe. Остальные файлы архива являются зашифрованными. Заметим, что у всех незашифрованных файлов несколько первых байт одинаковые (заголовок файла). Рассмотрим, например, первые 8 из них «0x4D 0x5A 0x90 0x00 0x03 0x00 0x00 0x00», что соответствует началу заголовка файла формата .EXE. Следовательно, можно предположить, что остальные файлы после их расшифрования должны содержать аналогичные 8 первых байт. Для определения ключа необходимо выполнить операцию «побитового исключающего ИЛИ» между первыми байтами любого из файлов Clockres.exe; plink.exe; pslist.exe и первыми байтами любого зашифрованного файла (например puttygen_enc.exe): 4D 5A 90 00 03 00 00 00 ^ FB C7 26 9D B5 9D B6 9D = B6 9D B6 9D B6 9D B6 9D.

Таким образом, в качестве ключа использовалась последовательность – B6 9D. Далее необходимо применить операцию «побитового исключающего ИЛИ» между всеми байтами зашифрованных файлов и байтами, полученными циклическим повторением последовательности байтов ключа, т.е. B6 9D B6 9D ... B6 9D, для чего необходимо написать программное средство, позволяющее выполнить данную операцию в автоматическом режиме. Результатом работы данного программного средства являются файлы формата .EXE, которые успешно запускаются.

На втором этапе необходимо определить файлы, содержащие сигнатуру «0x0A 0x0B 0x0C 0x0F»: clockres.exe, plink.exe, whois.exe, streams.exe, sync.exe, strings.exe. Следовательно, остальными программами школьник сможет воспользоваться, не отключая антивирус. Заметим, что файл whois_enc.exe содержал искомую сигнатуру только в зашифрованном виде.

Ответ: Все файлы, кроме clockres.exe, plink.exe, streams.exe, sync.exe, strings.exe.

Задача 5. Exploit

Имеется программа на языке C:

```
void main()
{
    char a[10];
    printf("Введите строку:");
    gets(a);
    printf("Вы ввели %d символов", strlen(a));
}
```

При компиляции получился следующий программный код:

Адрес памяти	Байты в памяти	Их содержание
230D123D	C2 E2 E5 E4 E8 F2 E5 20 F1 F2 F0 EE EA F3 00	Строка «Введите строку» с нулевым байтом в конце
230D124C	C2 FB 20 E2 E2 E5 EB E8 20 25 64 20 F1 E8 EC E2 EE EB EE E2 00	Строка «Вы ввели %d символов» с нулевым байтом в конце
230D1261	00 00 00 00 00 00 00 00 00 00	массив <i>a</i> (10 элементов типа char)
230D126B	68 3D 12 0D 23	Машинная команда, передающая адрес строки «Введите строку» (230D123D) в подпрограмму <i>printf</i> как параметр
230D1270	FF 15 14 72 0D 23	Машинная команда, вызывающая подпрограмму <i>printf</i> , расположенную по адресу 230D7214

230D1276	83 C4 04	Вспомогательная машинная команда, выполняемая после возвращения из подпрограммы <i>printf</i> , имеющей один параметр
230D1279	68 61 12 0D 23	Машинная команда, передающая параметр <i>a</i> , расположенный по адресу 230D1261, в подпрограмму <i>gets</i>
230D127E	FF 15 10 72 0D 23	Машинная команда, вызывающая подпрограмму <i>gets</i> , расположенную по адресу 230D7210
230D1284	83 C4 04	Вспомогательная машинная команда, выполняемая после возвращения из подпрограммы <i>gets</i> , имеющей один параметр
230D1287	68 61 12 0D 23	Машинная команда, передающая параметр <i>a</i> , расположенный по адресу 230D1261, в подпрограмму <i>strlen</i>
230D128C	E8 D5 FD FF FF	Машинная команда, вызывающая подпрограмму <i>strlen</i>
230D1291	83 C4 04	Вспомогательная машинная команда, выполняемая после возвращения из подпрограммы <i>strlen</i> , имеющей один параметр
230D1294	50	Машинная команда, передающая возвращаемое значение функции <i>strlen</i> в подпрограмму <i>printf</i> как параметр
230D1295	68 4C 12 0D 23	Машинная команда, передающая адрес строки « <i>Вы ввели %d символов</i> » (230D124C) в подпрограмму <i>printf</i> как параметр
230D129A	FF 15 14 72 0D 23	Машинная команда, вызывающая подпрограмму <i>printf</i> , расположенную по адресу 230D7214
230D12A0	83 C4 08	Вспомогательная машинная команда, выполняемая после возвращения из подпрограммы <i>printf</i> , имеющей два параметра
230D12A3	C3	Машинная команда, завершающая программу

Применяемая реализация подпрограммы *gets* принимает на вход любые данные без ограничений, в том числе специальные и непечатаемые символы (нулевой байт, символ конца строки и т.п.). Окончанием входного потока считается комбинация байтов «0x0D0A».

Укажите входную последовательность байтов (в шестнадцатеричном формате), после ввода которой программа выведет строку «*Информационная культура*».

Решение

Нетрудно видеть, что если передать на вход программы более 10 байт данных, эти данные переполнят буфер и перезапишут код выполняющейся программы. Фактически необходимо подобрать данные, которые при их размещении, начиная с адреса 230D1261, сформируют, начиная с адреса 230D1287 (следующая машинная команда после вызова *gets* и следующей за ним вспомогательной машинной команды), скомпилированный образ следующего оператора языка C:

```
printf («Информационная культура»);
```

за которым следует машинная команда, завершающая программу. В программе есть похожий оператор:

```
printf ("Введите строку:");
```

который компилируется в три машинные команды:

```
68 3D 12 0D 23
FF 15 14 72 0D 23
83 C4 04
```

Первая команда присутствует в программе в четырех экземплярах, между собой они различаются только вторым байтом, который во всех случаях совпадает с последним байтом числового значения передаваемого в подпрограмму адреса. Можно заметить, что структура команды следующая:

Код_команды (68) адрес_операнда (4 байта)

Для передачи в качестве параметра подпрограммы произвольного адреса (например, 12345678), нужно выполнить команду вида:

```
68 78 56 34 12
```

Подпрограмма *printf* вызывается в команде дважды, соответствующая команда имеет вид:

```
FF 15 14 72 0D 23
```

За ней должна следовать вспомогательная команда:

```
83 C4 04
```

И команда завершения программы:

```
C3
```

Итак, по адресу 100D1287 необходимо разместить следующие байты:

```
68 XX XX XX XX FF 15 14 72 0D 23 83 C4 04 C3
```

где XX XX XX XX – записанный в обратном порядке адрес строки «Информационная культура», которую надо разместить где-то в памяти атакуемой программы. Строка занимает 24 байта (вместе с завершающим нулевым байтом), в начале буфера вполне хватает места для нее, можно разместить эту строку прямо по адресу переменной *a*. Итоговый эксплойт может иметь, например, такой вид:

Адрес	Содержимое	Описание
	C8 ED F4 EE F0 EC E0 F6 E8 EE ED ED E0 FF 20 EA F3 EB FC F2 F3 F0 E0 00	Строка «Информационная культура»
	12 34 56 78 9A BC DE F0 12 34 56	Произвольные данные, которые никак не будут использоваться
	83 C4 04	Вспомогательная машинная команда, выполняемая после возвращения из подпрограммы <i>gets</i> , внутри которой произошло переполнение буфера
	68 61 12 0D 23	Машинная команда, передающая адрес строки «Информационная культура» в подпрограмму <i>printf</i> как параметр
	FF 15 14 72 0D 23	Машинная команда, вызывающая подпрограмму <i>printf</i> с одним параметром
	83 C4 04	Вспомогательная машинная команда, выполняемая после возвращения из подпрограммы <i>printf</i>
	C3	Машинная команда, завершающая программу