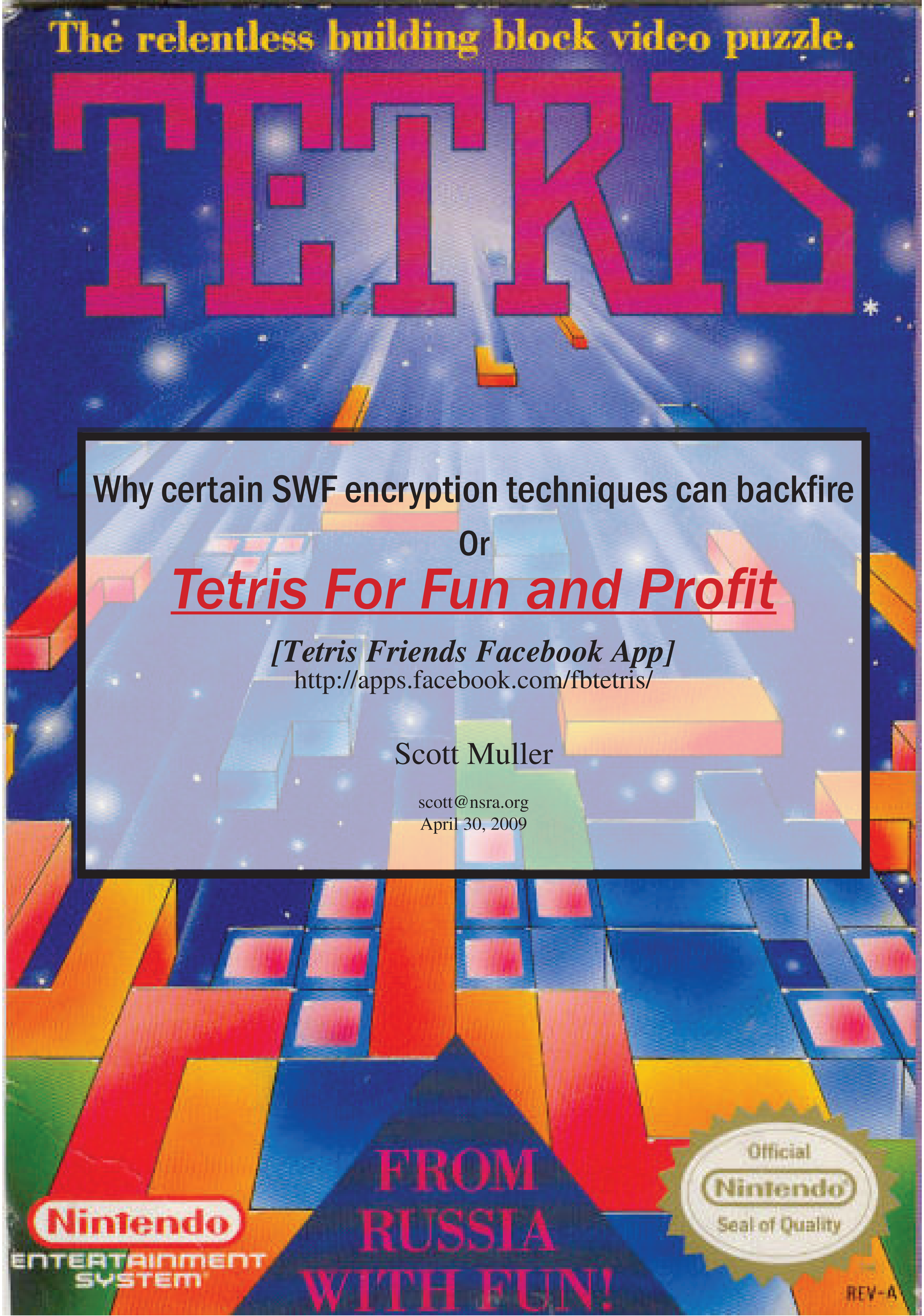


Why certain SWF encryption techniques can backfire

Or

Tetris For Fun and Profit

[Tetris Friends Facebook App]

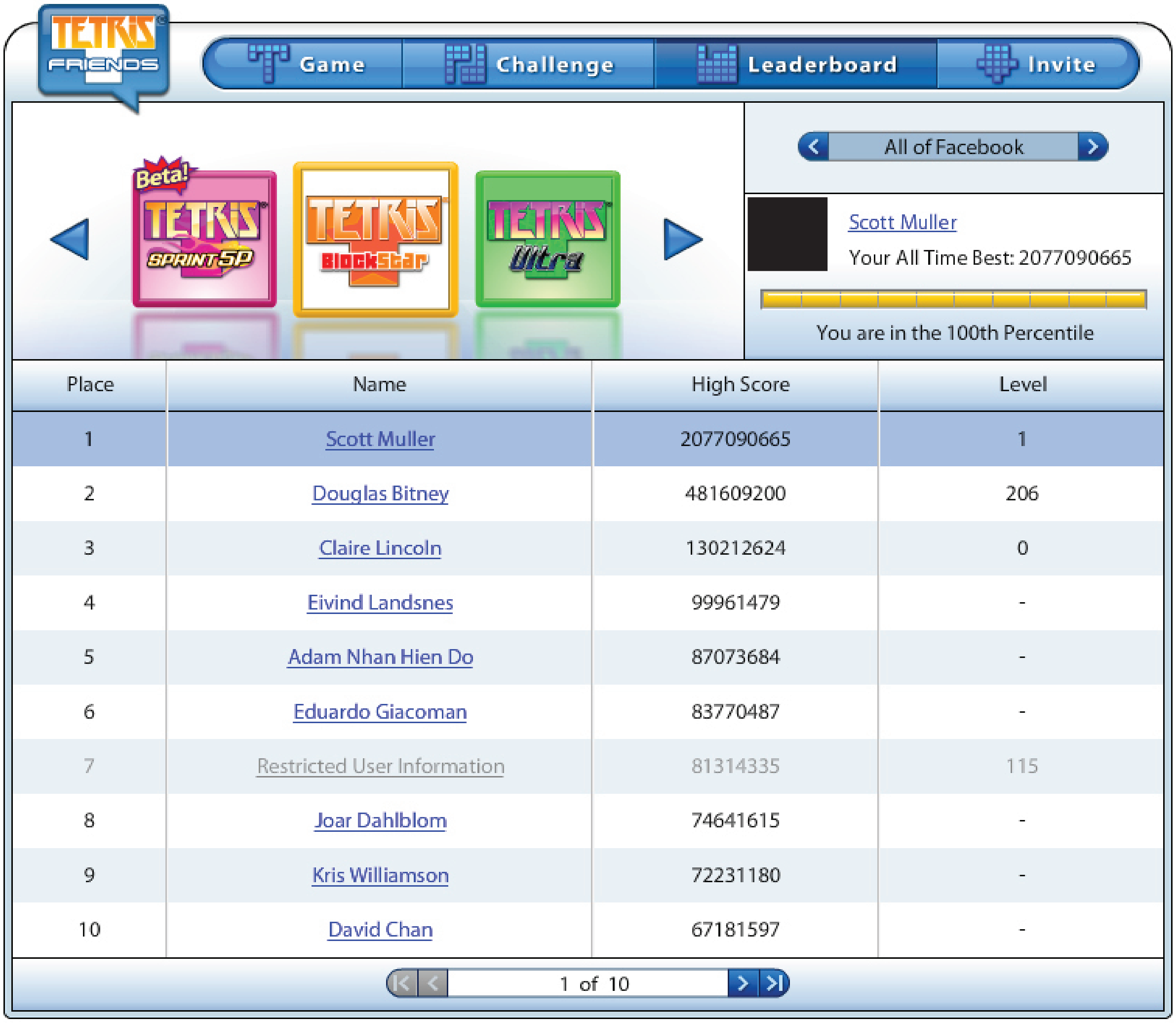
http://apps.facebook.com/fbtetris/

Scott Muller

scott@nsra.org

April 30, 2009

How could this happen?



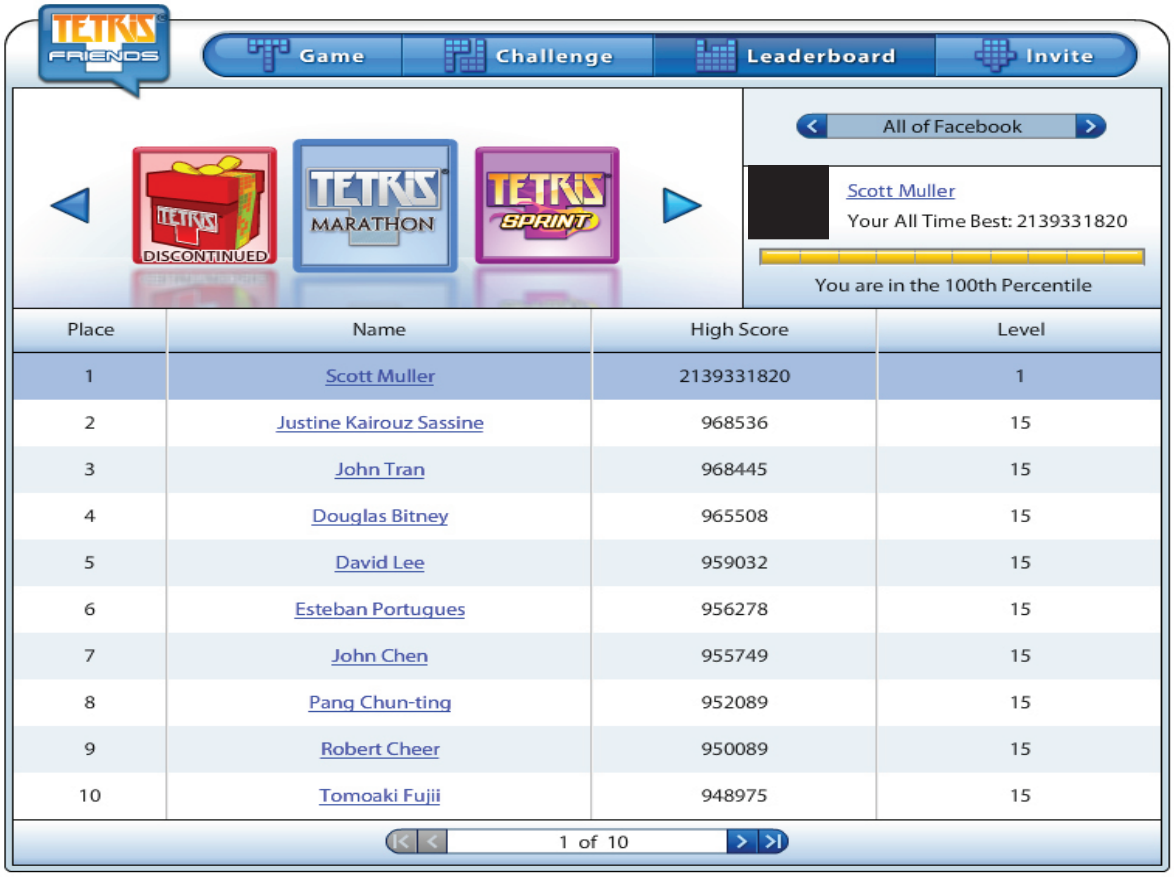
Scott Muller

Your All Time Best: 2077090665

You are in the 100th Percentile

Place	Name	High Score	Level
1	Scott Muller	2077090665	1
2	Douglas Bitney	481609200	206
3	Claire Lincoln	130212624	0
4	Eivind Landsnes	99961479	-
5	Adam Nhan Hien Do	87073684	-
6	Eduardo Giacoman	83770487	-
7	Restricted User Information	81314335	115
8	Joar Dahlblom	74641615	-
9	Kris Williamson	72231180	-
10	David Chan	67181597	-

1 of 10



Scott Muller

Your All Time Best: 2139331820

You are in the 100th Percentile

Place	Name	High Score	Level
1	Scott Muller	2139331820	1
2	Justine Kairouz Sassine	968536	15
3	John Tran	968445	15
4	Douglas Bitney	965508	15
5	David Lee	959032	15
6	Esteban Portugues	956278	15
7	John Chen	955749	15
8	Pang Chun-ting	952089	15
9	Robert Cheer	950089	15
10	Tomoaki Fujii	948975	15

1 of 10

I give them credit- they made a good effort.

They not only employed obfuscation techniques on the Flash code- but also employed a function that encrypts the memory based on a random number. The encryption makes it impossible to edit the numbers directly- or locate a known value in memory. Their obfuscation techniques confuse most disassemblers/decompilers and made a direct attack difficult (I tried FLASM, Trillix, Sothink)...

However, their obfuscation technique was their downfall.

Since a simple disassemble/reassemble approach, using decompiled ActionScript as a guide, is made difficult by the obfuscation they employed, the AVM (ActionScript Virtual Machine) bytecode will have to be found someother way- and parsed and edited directly.

Sounds like fun!

Most of the bytecode could be deciphered using swfdump. Reviewing the code, it could be seen that they were employing a common technique to hide their variables- such as well as the as3crypto library.

Cute idea, but it means nothing if we manipulate the variable *before* it is passed to the encryption routine.

A conveniently named function, *incrementValue* was discovered.

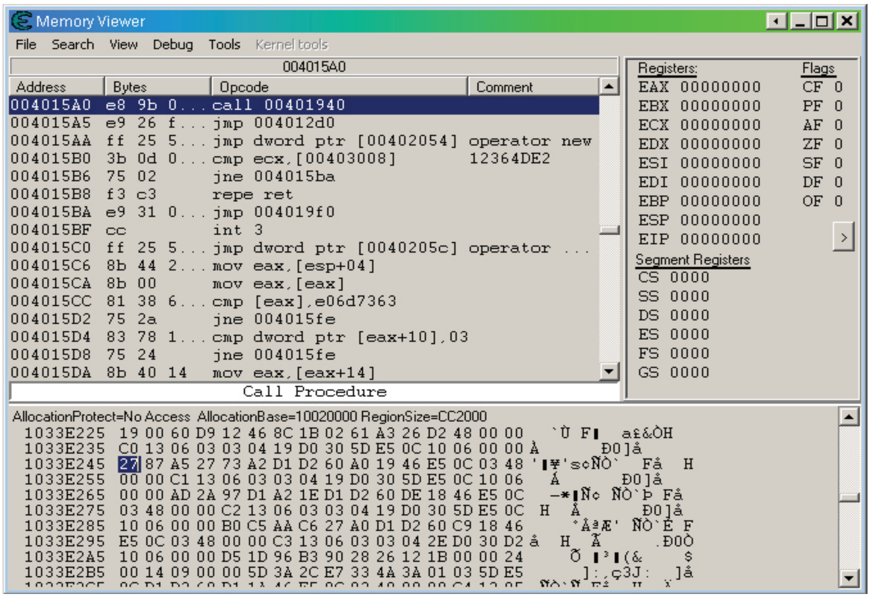
Low and behold, their obfuscator made a nice little area for us to insert a few bytes before the unencrypted value is sent to the encrypted memory storage function. Because one of the techniques their obfuscator uses is to insert random jumps in the code, from the addresses 00003->00009 in the function is fair game.

But how to hijack the program, and with what?

One fun thing about Flash running in Firefox is that the AVM bytecode is stored in memory when the SWF file is instantiated. This means, we can rewrite the actual bytecode in real-time.

```
slot 5: var <q>[private]::crypto:<q>[public]TetrionShell.util::TEA = null
   final  method <q>[public]::Number <q>[public]::incrementValue=(<q>[public]::String,
<q>[public]::Number)(2 params, 0 optional)
   [stack:6 locals:3 scope:3-4 flags:] slot:5

   {
     00000) + 0:0          getlocal_0
     00001) + 1:0          pushscope
     00002) + 0:1          findpropstrict
                           <q>[private]::processValue
     00003) + 1:1          jump ->10
     00004) | 0:0          pushfalse
     00005) | 0:0          astypelate
     00006) | 0:0          lshift
     00007) | 0:0          pushfalse
     00008) | 0:0          convert_i
     00009) | 0:0          multiply
     00010) + 1:1          getlocal_1
     00011) + 2:1          getlocal_2
     00012) + 3:1          getlex <q>[private]::
                           INCREMENT
     00013) + 4:1          callproperty
                           <q>[private]::processValue,
                           3 params
     00014) + 1:1          returnvalue
   }
```



```
004015A0

004015A0  e8 9b 0...call 00401940
004015A5  e9 26 f...jmp 004012d0
004015AA  ff 25 5...jmp dword ptr [00402054] operator new
004015B0  3b 0d 0...cmp ecx,[00403008]        12364DE2
004015B6  75 02     jne 004015ba
004015B8  f3 c3     repe ret
004015BA  e9 31 0...jmp 004019f0
004015BF  cc        int 3
004015C0  ff 25 5...jmp dword ptr [0040205c] operator ...
004015C6  8b 44 2...mov eax,[esp+04]
004015CA  8b 00     mov eax,[eax]
004015CC  81 38 6...cmp [eax],e06d7363
004015D2  75 2a     jne 004015fe
004015D4  83 78 1...cmp dword ptr [eax+10],03
004015D8  75 24     jne 004015fe
004015DA  8b 40 14  mov eax,[eax+14]
                    Call Procedure

EAX 00000000     CF 0
EBX 00000000     PF 0
ECX 00000000     AF 0
EDX 00000000     ZF 0
ESI 00000000     SF 0
EDI 00000000     DF 0
EBP 00000000     OF 0
ESP 00000000
EIP 00000000

CS 0000
SS 0000
DS 0000
ES 0000
FS 0000
GS 0000

AllocationProtect=No Access  AllocationBase=10020000 RegionSize=CC2000
 1033E225 19 00 60 D9 12 46 8C 1B 02 61 A3 26 D2 48 00 00   `Ù F▌  a£&ÒH
 1033E235 C0 13 06 03 03 04 19 D0 30 5D E5 0C 10 06 00 00 À       Ð0]å
 1033E245 27 87 A5 27 73 A2 D1 D2 60 A0 19 46 E5 0C 03 48 '▌¥'s¢ÑÒ`  Få  H
 1033E255 00 00 C1 13 06 03 03 04 19 D0 30 5D E5 0C 10 06   Á       Ð0]å
 1033E265 00 00 AD 2A 97 D1 A2 1E D1 D2 60 DE 18 46 E5 0C   -*▌Ñ¢ ÑÒ`Þ Få
 1033E275 03 48 00 00 C2 13 06 03 03 04 19 D0 30 5D E5 0C  H  Â       Ð0]å
 1033E285 10 06 00 00 B0 C5 AA C6 27 A0 D1 D2 60 C9 18 46     °ÅªÆ' ÑÒ`É F
 1033E295 E5 0C 03 48 00 00 C3 13 06 03 03 04 2E D0 30 D2 å  H  Ã      .Ð0Ò
 1033E2A5 10 06 00 00 D5 1D 96 B3 90 28 26 12 1B 00 00 24     Õ ▌³▌(&    $
 1033E2B5 00 14 09 00 00 5D 3A 2C E7 33 4A 3A 01 03 5D E5      ]:,ç3J:  ]å
```

Locating the junk bytes that was jumped over by their obfuscation technique in the Firefox memory was fairy easy, using CheatEngine:

```
[27 87 A5 27 73 A2]
```

OK- Now what to put there...

Looking at the code, [local variable 2] is used to store the value to increment as an integer... I also found out that the function is used even if a block is simply sped down using the down arrow (usually worth 2 points)... Inserting a massive number into the variable should make this as easy as hitting one button. Since integers are referenced using an index, we need to find a reference to an integer in the running bytecode that we can use...

Searching the code, one (and a fairly blatantly large one) was found:

pushint 1322485955

Finding the index to it was made fairly easy by simply searching the Firefox process memory for the neighboring junk code- which is usually single byte op codes (thanks again for using that obfuscator!) :

```
[D5 04]
```

Now to write some bytecode to replace lines 00003->00009, overriding the jump (again- thanks for obfuscating):

```
2D D5 04 63 02 (02 02....):

pushint         1322485955[D5 04]
setlocal        register 2
nop, nop...
```

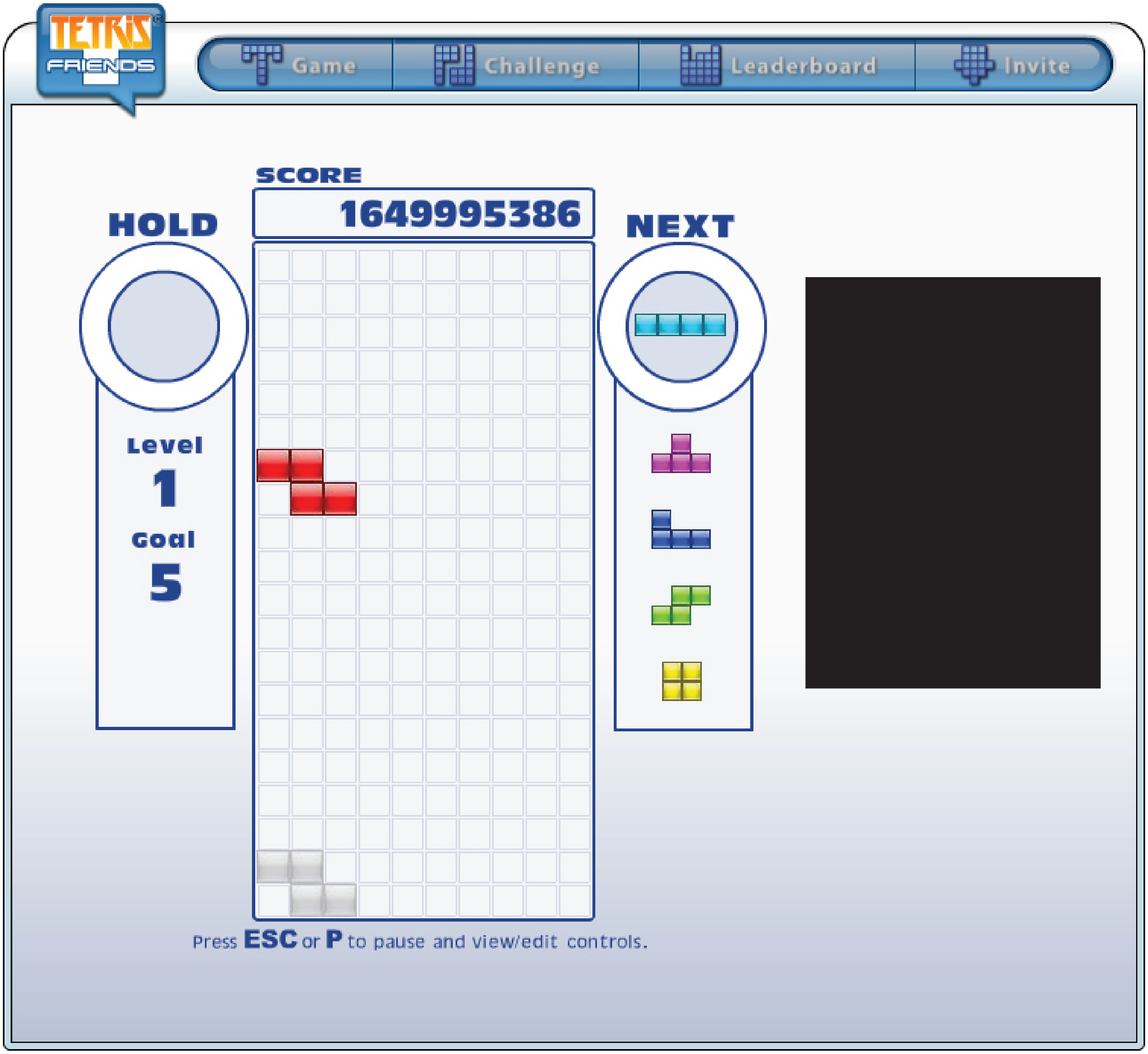
That should do the trick.

OK- Now let’s play a game and see how the function performs...

```
method <q>[public]::void
<q>[public]::Create=(<q>[public]flash.
display::Sprite, <q>[public]::Function)(2
params, 0 optional)
   [stack:5 locals:4 scope:4-5 flags:] slot:0

  {
    00000) + 0:0        getlocal_0
    00001) + 1:0        pushscope
    00002) + 0:1        pushnull
    00003) + 1:1        jump ->10
    00004) | 0:0        instanceof
    00005) | 0:0        divide
    00006) | 0:0        not
    00007) | 0:0        subtract
    00008) | 0:0        checkfilter
    00009) | 0:0        rshift
    00010) + 1:1        coerce_s
    00011) + 1:1        setlocal_3
    00012) + 0:1        pushfalse
    00013) + 1:1        iftrue ->134
    00014) + 0:1        getlocal_0
    00015) + 1:1        getlocal_2
    00016) + 2:1        setproperty <q>[private]::mLoginRs
    00017) + 0:1        pushtrue
    00018) + 1:1        iffalse ->122
    00019) + 0:1        getlocal_0
    00020) + 1:1        findpropstrict <q>[public]
                        Entities::UserLite
    00021) + 2:1        constructprop <q>[public]
                        Entities::UserLite, 0 params
    00022) + 2:1        setproperty <q>[private]::mUserLite
    00023) + 0:1        pushtrue
    00024) + 1:1        iffalse ->116
    00025) + 0:1        getlocal_0
    00026) + 1:1        getproperty <q>[private]::mUserLite
    00027) + 1:1        pushbyte 3
    00028) + 2:1        setproperty <q>[public]::mId
    00029) + 0:1        pushtrue
    00030) + 1:1        iffalse ->113
    00031) + 0:1        getlocal_0
    00032) + 1:1        getproperty <q>[private]::mUserLite
    00033) + 1:1        pushint 1322485955
    00034) + 2:1        setproperty <q>[public]::mUid
    00035) + 0:1        pushtrue
    00036)               .....................
```

AND BOOM GOES THE DYNAMITE!



HOLD

SCORE
1649995386

NEXT

Level
1
Goal
5

Press **ESC** or **P** to pause and view/edit controls.

References:

AVM2 Bytecode Reference:

http://www.anotherbigidea.com/javaswf/avm2/AVM2Instructions.html

CheatEngine:

http://www.cheatengine.org/

(Linux) FLASM:

http://www.nowrap.de/flasm.html

(Linux) Swf Tools (swfdump):

http://www.swftools.org/

as3crypto

http://code.google.com/p/as3crypto/

Memory encryption technique

http://mikegrundvig.blogspot.com/2007/05/encrypting-variables-in-memory-to.html

scott@nsra.org