Chewing on the 0xDEADBEEF
(redacted 26th March ‘08)

Mike Bond
10th August ‘07
Why redact this talk?

- Surely the crooks know this stuff already? Well maybe not, this is not crooks we are up against, just amateur hackers who want to cheat at games (little money in cheating at FPS yet), and its probably not illegal.
- True, Joint Ops is already suffering from cheats, and almost dead, but I don’t want to be putting the final nail in the coffin on what was an exceptionally good game in its time.
- Another 6 months or so and there should be no harm in releasing full detail of this talk.
- Some redaction in the screenshots was done to protect privacy of testers (and unwitting testers)
- Any bona fide researchers in game cheating/network effects are welcome to take a copy of the full talk, plus source code etc, so long as they can satisfy me it will be put to good use. Email Mike.Bond@cl.cam.ac.uk, Phone +44 7890 171913
The 0xDEADBEF

```
.text:0041E908  .globl _start  ; CODE XREF: C76_LOGGING_BRENDA_s
.text:0041E909  .type  _start, %function
.text:0041E90B  _start:
.text:0041E90C  .cfi_startproc
.text:0041E90D  mov  ebp, esp
.text:0041E90E  cmp  ebp+84h, 0
.text:0041E90F  jz   loc_41E917
.text:0041E910  test  byte ptr [ebp+84h], 7
.text:0041E911  jz   loc_41E930
.text:0041E912  cmp  byte ptr [ebp+84h], 0
.text:0041E913  jnz  loc_41E930
.text:0041E914  cmp  byte ptr [ebp+84h], 0
.text:0041E915  je   loc_41E930
.text:0041E916  push  ebx
.text:0041E917  lea  ebx, [ebp+84h]
.text:0041E918  jz   no_overrun_or_underrun
.text:0041E919  mov  eax, 0xDEADBEF
.text:0041E91A  cmp  [esi+30h], eax
.text:0041E91B  jz   short da_check_For_overrun
.text:0041E91C  memory_buffer_underrun:
.text:0041E91D  lea  eax, [esp+410h+var_404_string]
.text:0041E91E  push  eax
.text:0041E91F  mov  byte ptr [ebp+08h], 1
.text:0041E920  call  892_c92_checkyloggy_sub_42C850
.text:0041E921  mov  edx, [esi+0CH]
.text:0041E922  lea  eax, [esi+edx+38h]
.text:0041E923  mov  edx, [esi+14h]
.text:0041E924  lea  ecx, [esi+34h]
.text:0041E925  push  ecx
.text:0041E926  push  eax
.text:0041E927  push  edx
.text:0041E928  lea   eax, [edi+1023]
.text:0041E929  cdq
.text:0041E92A  and  edx, 1023
.text:0041E92B  add  eax, edx
.text:0041E92C  sar  eax, 10
.text:0041E92D  push  eax
.text:0041E92E  push  eax
.text:0041E92F  push  eax
.text:0041E930  mov  eax, [esi+140h] ; paranoid 512K (kilobytes)
.text:0041E931  .cfi_endproc
```
Contents

This work discusses reverse engineering and cryptanalysis of the encrypted data packets transmitted by an online multiplayer tactical shooter computer game “Joint Operations”.

• How it does the encryption
• How I cracked it
• Sturgeon’s Razor
• What if?
• Future Work
Packet “Encryption” Overview
Henry Decrypt

Packets longer than X bytes encoded as follows:

Packets less than X bytes with X
Main Decrypt Routine

- Takes null terminated array (usually ASCII string) as key input
- Always discards first byte without use
- Consists of key schedule derivation plus four base routines, all operating with byte-wise modulo addition
  - first quadratic equation
  - second quadratic equation
  - conditional string reverse (diffusion!)
  - Vigenère cipher
decrypt_mask2

void decrypt_mask2(ref byte[] message, UInt32 len, ref UInt32[] keyschedule)
{
    <snip>
}

nb. code looks weird because it is trans-literated from dissassembly
void decrypt_mask1(ref byte[] message, UInt32 len, UInt32 magic, UInt32 k3) {
    <snip>
}
decrypt_loopy

void decloopy(ref byte[] message, UInt32 len, byte[] key)
{
    <snip>

}
Three Doors

Behind one is a car, behind one freedom, and behind the other, certain death! Pick a door. Now I take away a door. Do you change your mind, or stick with your door?

Cryptanalysis

Static Reverse Engineering

Debugging
How I cracked it

1. intercepted packets using Ethereal: noted apparent encryption
2. created chosen chat messages, analysed by length of packet: detected individual packets corresponding to chat message
3. took differential between chosen plaintexts ‘aaaaaaaaaaaaa’ and ‘bbbbbbbbbbb’, looking for evidence of stream cipher
4. stream cipher theory validated, began reverse engineering to locate stream cipher
How I cracked it (2)

5. no evidence of stream cipher from examining XOR calls
6. worked upwards from the UDP sendto system call, found static hard-coded keys at 6 layers up (later henry at 3 layers also)
7. from static keys located crypto, discovered stream combined using byte-level addition
8. reverse engineered crypto algorithms (but not their calling structure)
How I cracked it (3)

9. studied packet ciphertexts and differentials between packets looking for evidence of crypto algorithm identified
10. found good evidence, but also evidence of another algorithm
11. went back looking and found “henry”
12. implemented decryption of henry
13. implemented decryption using static hard-coded key; by luck applying XXX yielded success.. a low entropy header
14. analysed packets looking for size and meaning of header
15. analysed differentials after first decrypt, looking for second decrypt
16. after much thought concluded second decrypt was indeed same algorithm, starting with XXX
How I cracked it (5)

17. Upgraded analysis tool to crack final key using chosen plaintext from chat messages.

18. Cracking algorithm uses brute-force to crack quadratic-equation based keys.

19. Calculates optimal Vignere cipher key using a tuned fitness function.
My Analysis Tool
My Analysis Tool (2)
What if?

• Suppose they’d used stronger key stream generator based on proper crypto algorithm (e.g. 3DES)?
  – Easier. Only need to reverse engineer to identify algorithm, not to re-implement
  – Easier. Crypto with proper diffusion characteristics makes it easier to determine when you have got it right. Still stuck with 95% accurate crypto reimplementations which occasionally get a byte wrong
  – Easier to locate in disassembly. Look for crypto-code characteristics

• Suppose they changed key every packet instead of every session?
  – Much easier to exhaust key space of weak cipher by sending a repeated message under many different keys
Sturgeon’s Razor

• My previous reverse-engineering rules
  – do what you can
  – give everything a name

• Occam and Sturgeon together…
  – “90% of everything is crap”
  – “All things being equal, the simplest explanation tends to be the best one.”

• Yields the new rules (used when explaining weird function behaviour):
  – “the simplest explanation is that it’s just a load of crap”
Future Work

• This work just a pre-requisite to experiments on “Neo-Tactics”

• I want to find out how the update rates of players in-game vary. You have 150 players in game, but you only have 600 bytes of data in your packet. What do you send?
  – Does perception of unfairness/cheating correspond to real bandwidth problems, or inadequacies/anomalies in use of available bandwidth?

• This work will also enable (study of) many sorts of undetectable cheating based on packet interception/analysis/rewriting
  – What sorts of hack are achievable if you can mess with the packets in tactical shooters? I hope to make a taxonomy.
Working Lag Analysis Tool

Hi res picture: see http://www.cl.cam.ac.uk/~mkb23/jopsdec/lag1-censor.png
Working Lag Analysis Tool (2)

Hi res picture: see http://www.cl.cam.ac.uk/~mkb23/jopsdec/leg3-censor.png