Out-of-Band File Transfer on Closed Networks

An Insider's Options

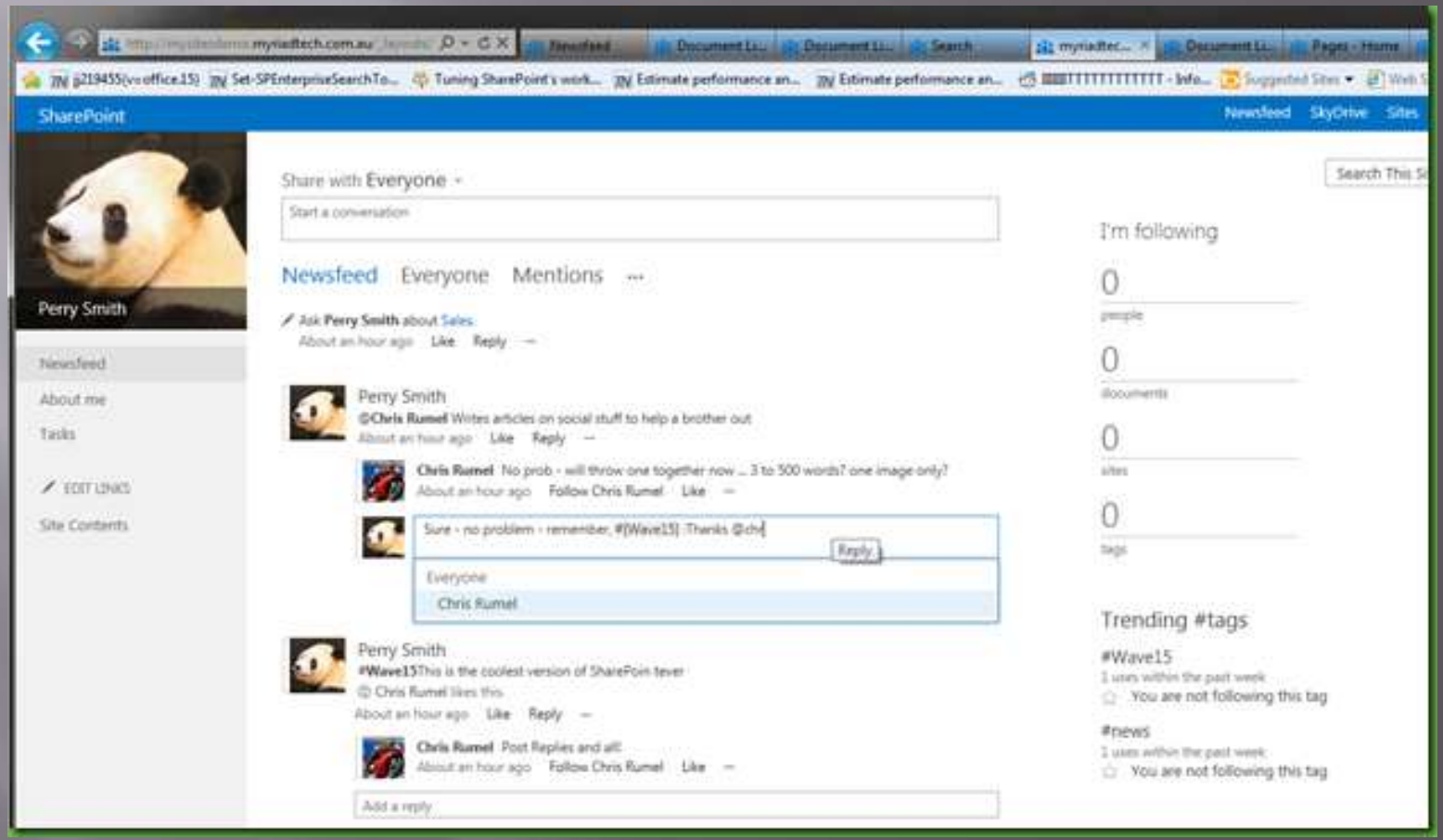
Michael Rich
mike@tofet.net

Overview

- The Challenge
- The Tools Available
- Phase 0 – Set up
- Phase 1 – Hex Attack
- Phase 2 – Attack of the Big Barcode
- Bringing it all Together
- Future/Branch Research Paths
- Conclusion

The Challenge

- There I was, hacking the collaboration portal..



The Challenge Tumble

- How could I intercept the POST call to modify the inputs?
  - Tamper Data, Burp Suite, etc..
- How could I forge the POST call?
  - Curl, wget, etc..
- Eventually: "How could I load one of these tools on to my closed, secure network without getting caught?"

The Conditions

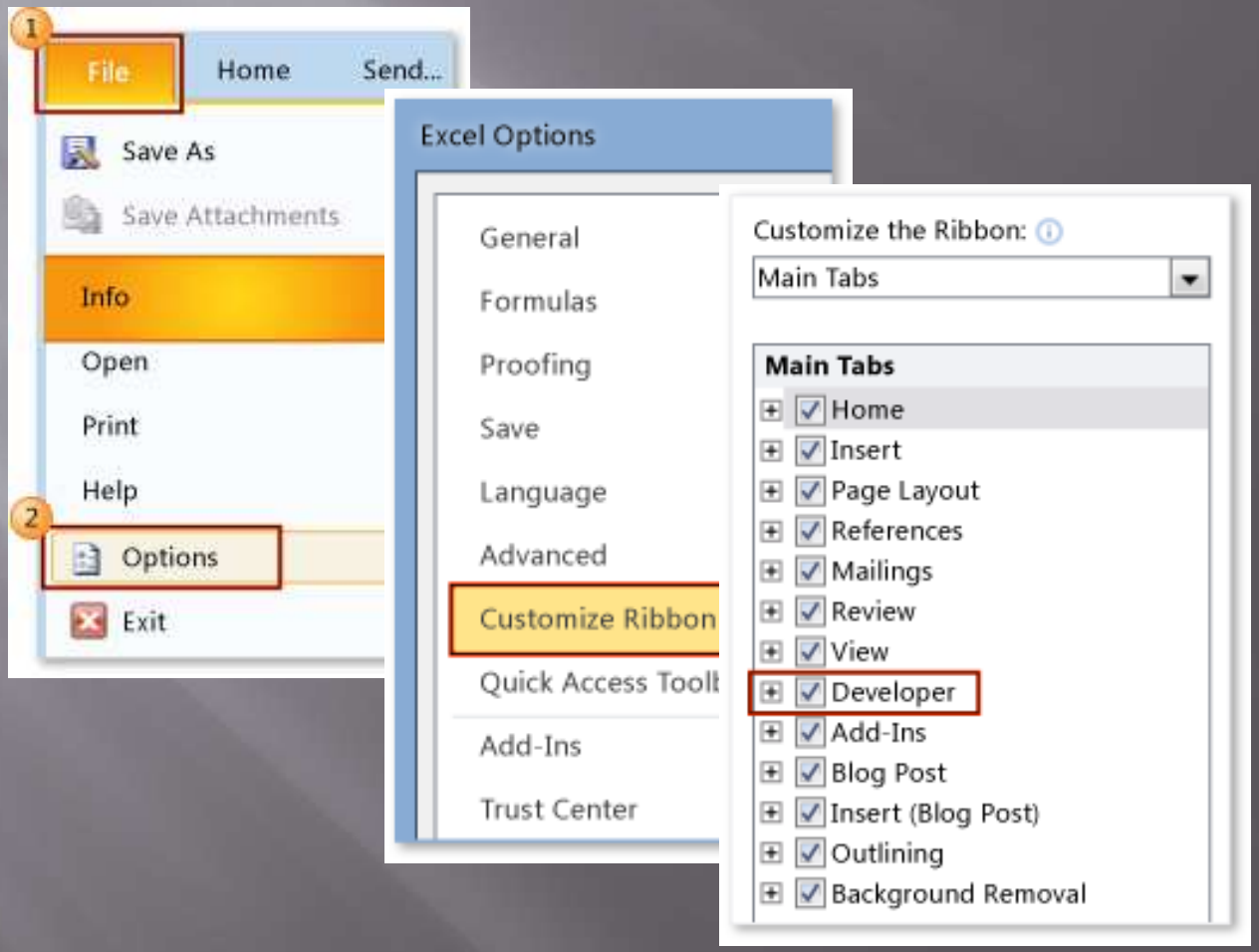
- Closed, secured network (sort of)
- USB ports secured & monitored
- CD use secured & monitored
- Host-based security system
- Data transfer entry points do exist (DOTS)
  - Not in control of attacker
  - Unknown scanning rules
  - Leaves logs
- Windows / MS-Office environment

Tools Available

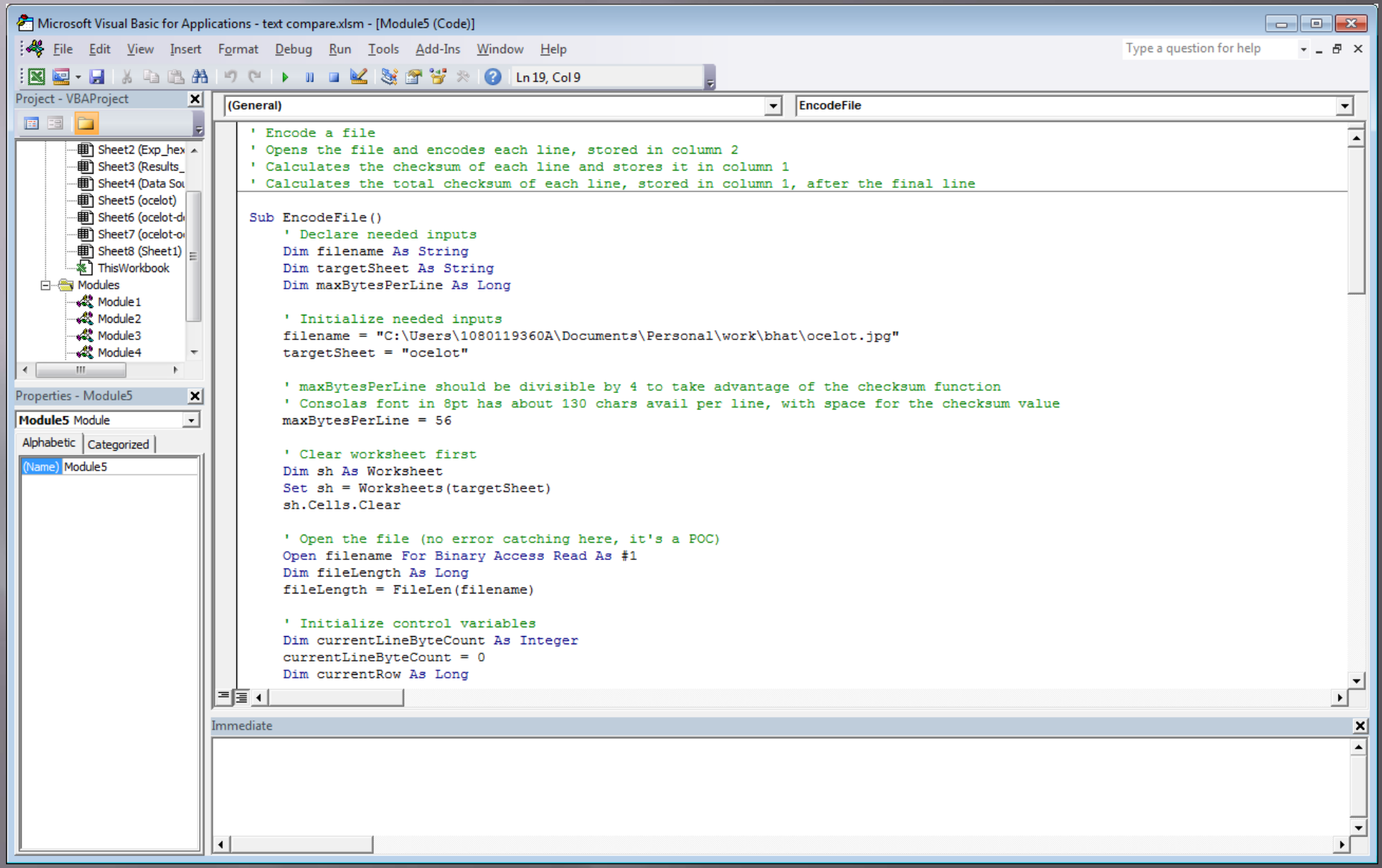
- MS Office = Visual Basic for Applications
- Professional level printers & scanners
- Adobe Acrobat OCR

Phase 0 – Set up

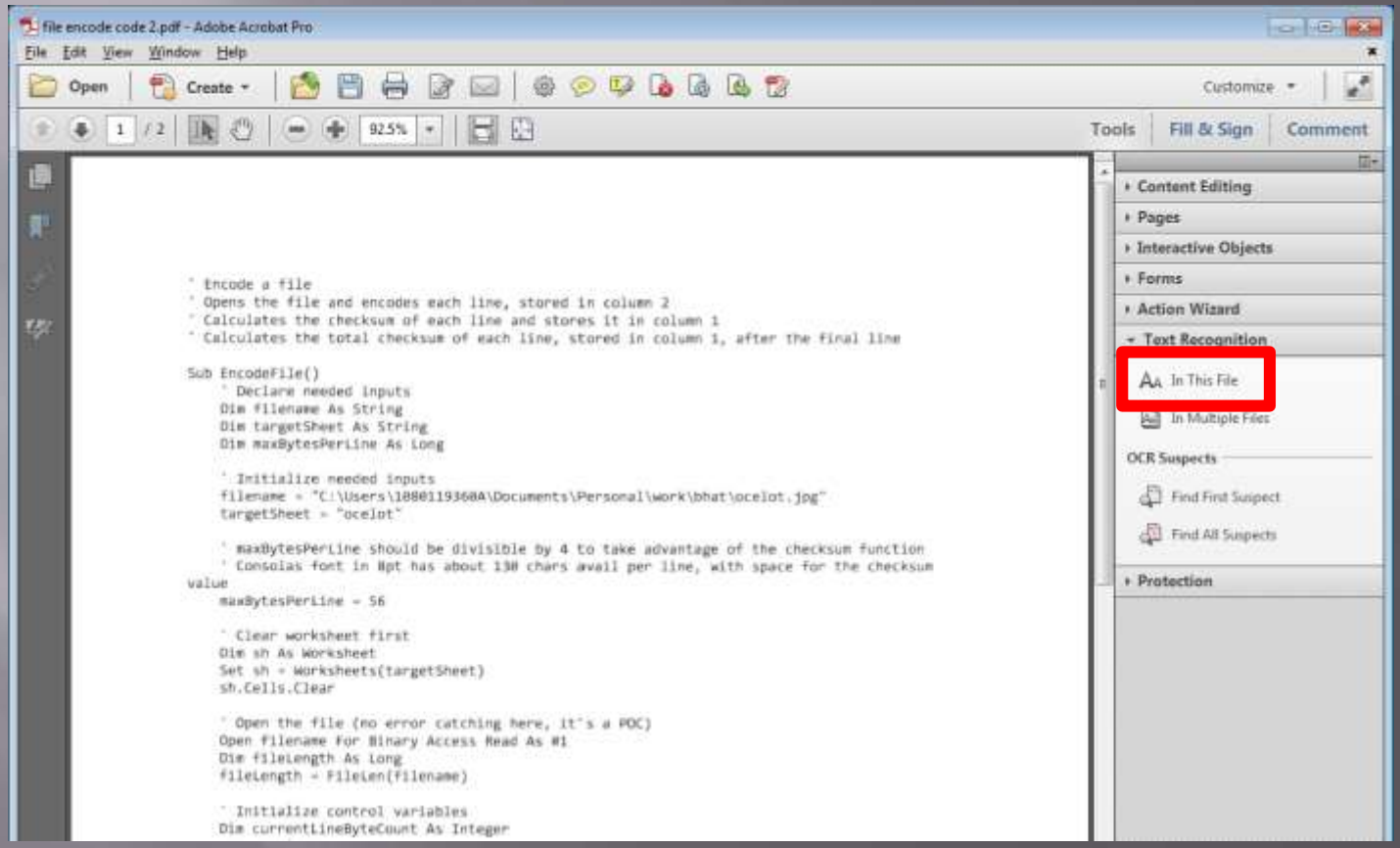
- Put Excel into attack mode



Phase 0 – Set up

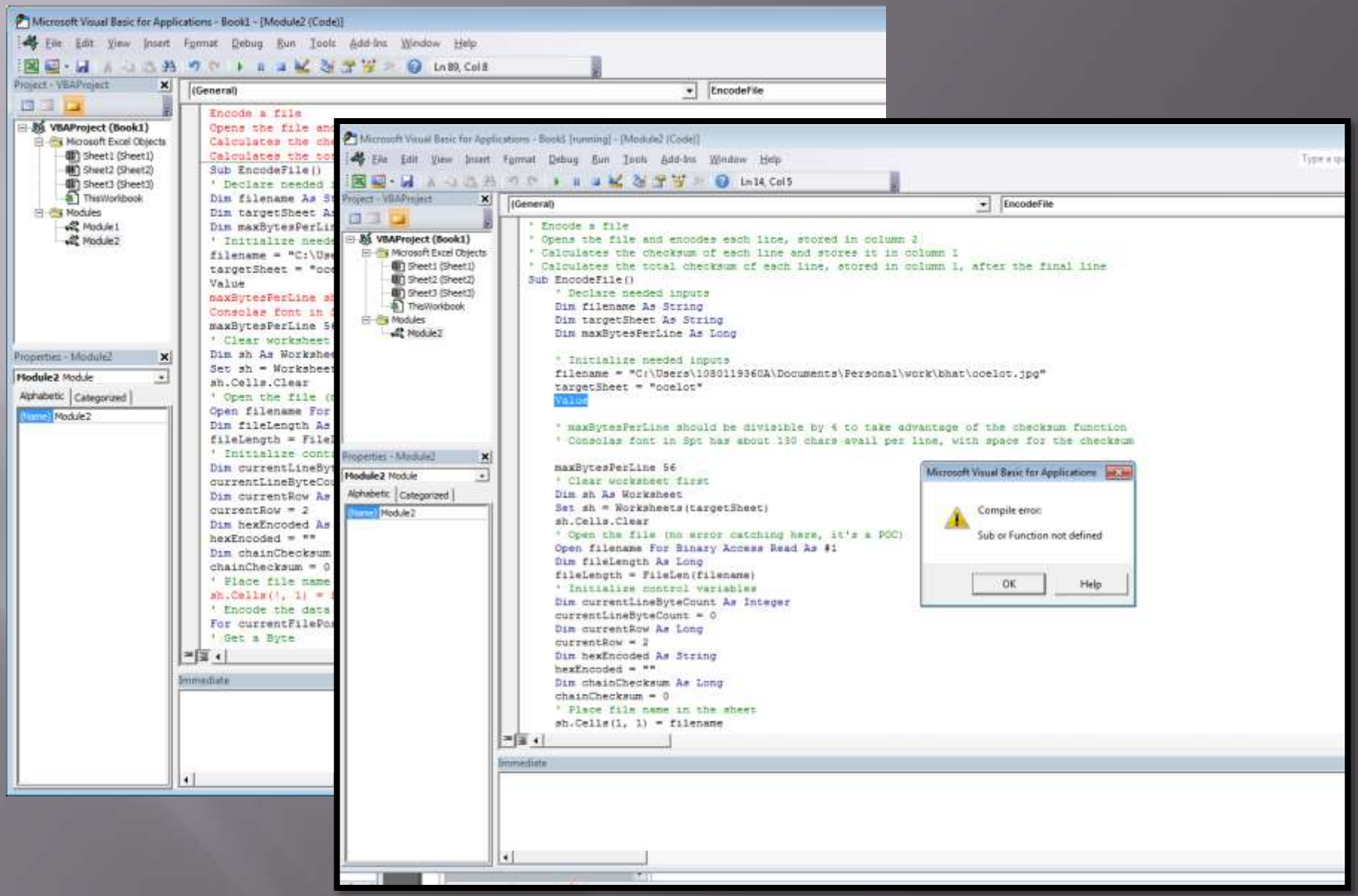


Phase 0 – Set up



- Consolas Font – Down to 8 pt font

Phase 0 – Set up

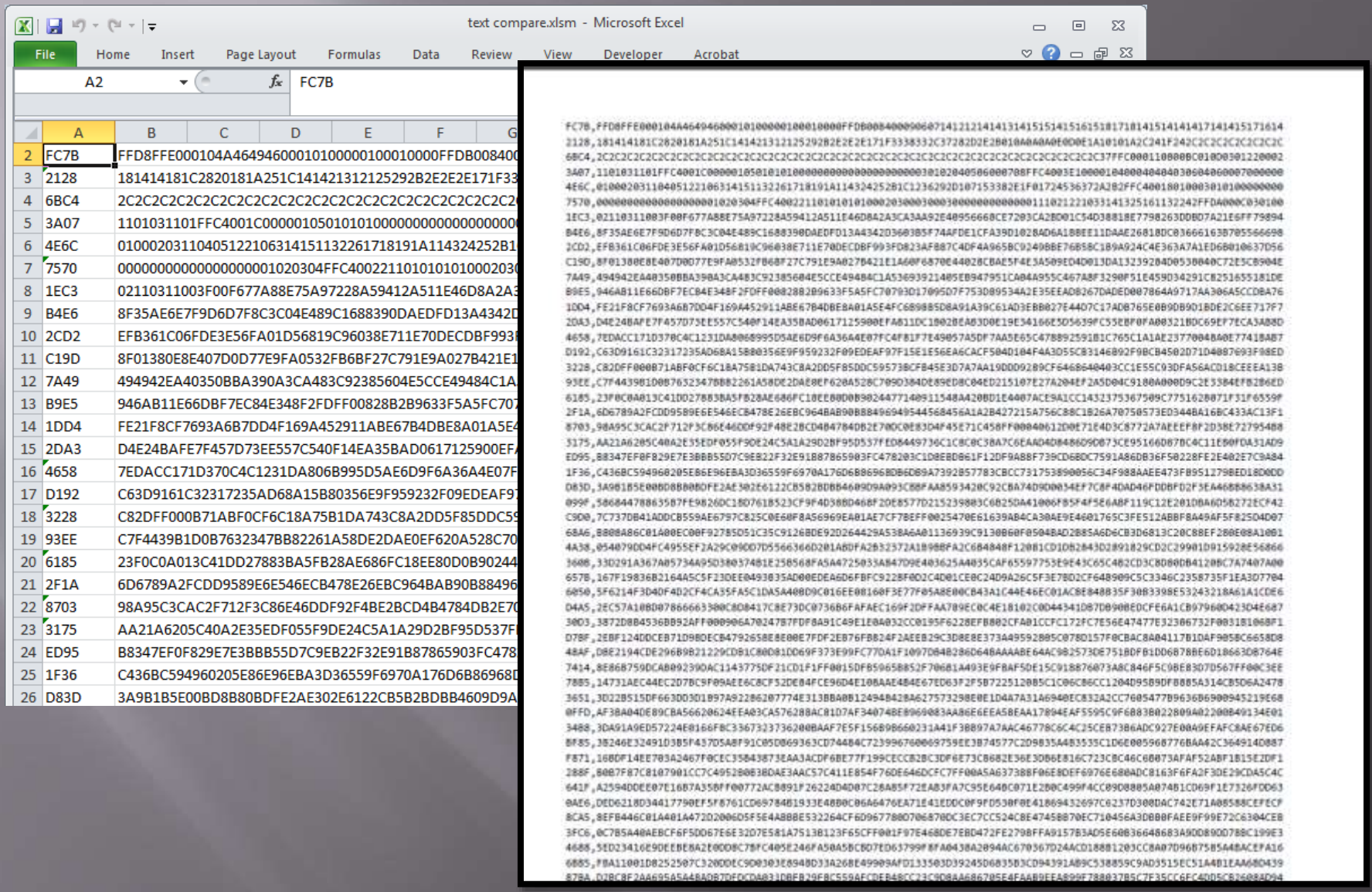


Phase 1 – Hex Attack

- Use Phase 0 methods to make Excel a binary file hex encoder/decoder
- Why hex?
  - Printable text
  - Tests showed excellent OCR results

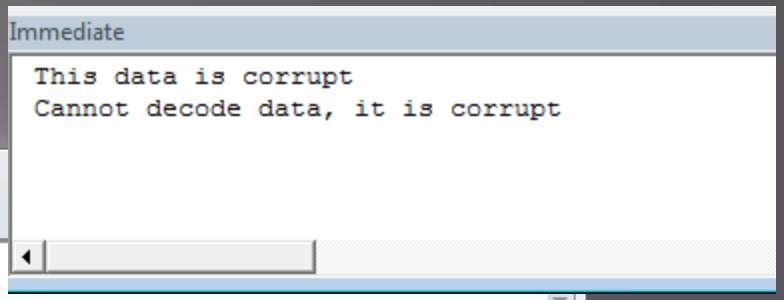
	Hex Encoding	Base64 Encoding
Encoding	Consolas, 8 pt font	Consolas, 12 pt font
Word Length Errors (80 char words)	0 in 73 words	9 in 73 words
Transcription Errors	0 in 5840 symbols	216 in 5840 symbols
Error Types	1 to 1	Many to Many

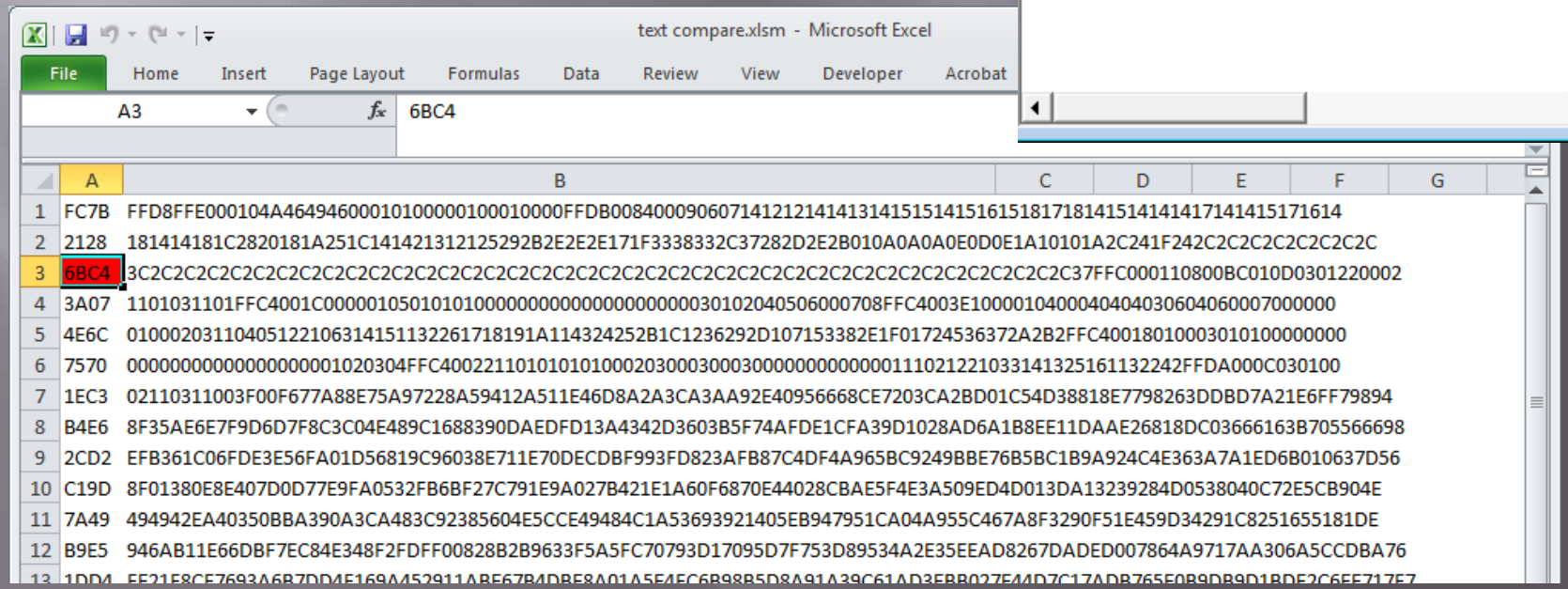
Phase 1 - Hex Attack



Phase 1 – Hex Attack

- Hex Encoding is good, but probably not perfect
- Need compact error detection
- 2-byte XOR checksum





	A	B
1	FC7B	FFD8FFE000104A4649460001010000010001000FFDB00840009060714121214141314151514151615181718141514141417141415171614
2	2128	181414181C2820181A251C141421312125292B2E2E2E171F3338332C37282D2E2B010A0A0A0E0D0E1A10101A2C241F242C2C2C2C2C2C2C2C
3	6BC4	3C2C2C2C2C2C2C2C2C2C2C2C2C2C2C2C2C2C2C2C2C2C2C2C2C2C2C2C2C2C2C2C2C2C2C2C2C2C2C2C2C2C2C2C2C37FFC000110800BC010D0301220002
4	3A07	1101031101FFC4001C0000010501010100000000000000000000030102040506000708FFC4003E10000104000404040306040600070000000
5	4E6C	0100020311040512210631415113226171819 1A114324252B1C1236292D107153382E1F01724536372A2B2FFC400180100030101000000000
6	7570	00000000000000000001020304FFC400221101010101000203000300030000000000000111021221033141325161132242FFDA000C030100
7	1EC3	02110311003F00F677A88E75A97228A59412A511E46D8A2A3CA3AA92E40956668CE7203CA2BD01C54D38818E7798263DDBD7A21E6FF79894
8	B4E6	8F35AE6E7F9D6D7F8C3C04E489C1688390DAEDFD13A4342D3603B5F74AFDE1CFA39D1028AD6A1B8EE11DAAE26818DC03666163B705566698
9	2CD2	EFB361C06FDE3E56FA01D56819C96038E711E70DECDBF993FD823AFB87C4DF4A965BC9249BBE76B5BC1B9A924C4E363A7A1ED6B010637D56
10	C19D	8F01380E8E407D0D77E9FA0532FB6BF27C791E9A027B421E1A60F6870E44028CBAE5F4E3A509ED4D013DA13239284D0538040C72E5CB904E
11	7A49	494942EA40350BBA390A3CA483C92385604E5CCE49484C1A53693921405EB947951CA04A955C467A8F3290F51E459D34291C8251655181DE
12	B9E5	946AB11E66DBF7EC84E348F2FDFF00828B2B9633F5A5FC70793D17095D7F753D89534A2E35EEAD8267DADED007864A9717AA306A5CCDBA76
13	1DD4	FF21E8CE7693A6B7DD4E169A452911ABE67B4DBE8A01A5E4EC6B98B5D8A91A39C61AD3EBB027E44D7C17ADB765E0B9DB9D1BDE2C6EE717E7

Phase I – Oops.. Real World

- Assumptions, assumptions, assumptions
- 1551 errors in 135,420 symbols (1.1 % error)
  - B to 8: 261; 1 to l: 359; 5 to S: 864
  - D to 0, O: 57; 6 to G,q,b: 3
- Alternative characters:
  - # for B
  - ? for D
- Auto-replace other major errors
  - “l”, “S”, “.”, “ “
- Add strong visual indicators
- 1 manual correction in 1210 lines of text

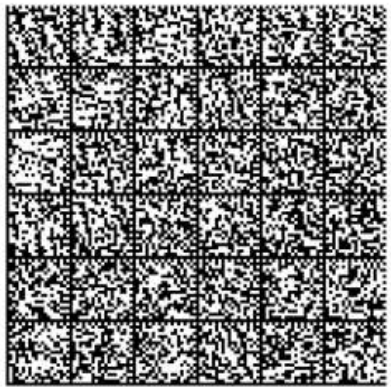
Demo Time!

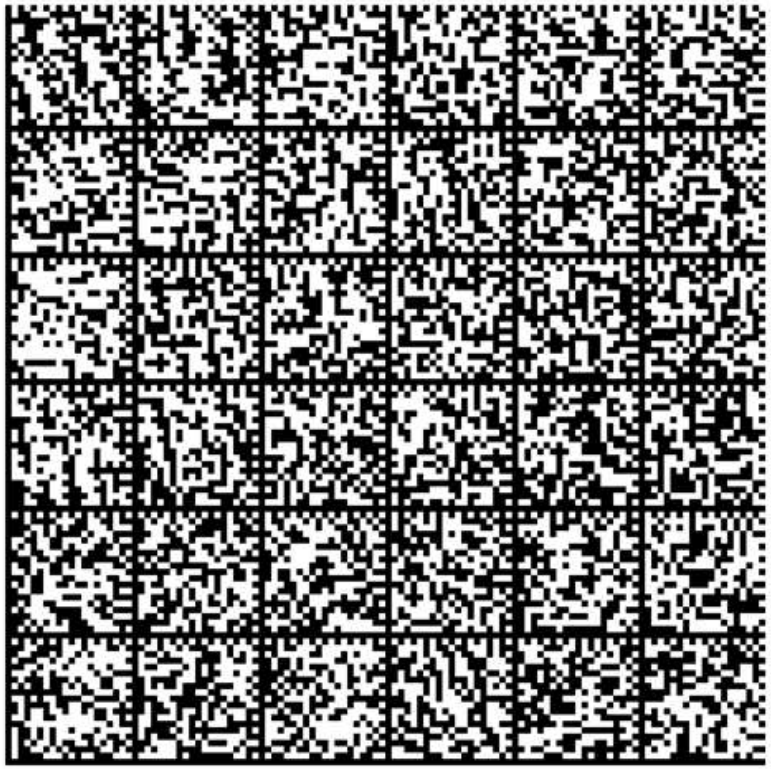
Phase 1 – Hex Attack

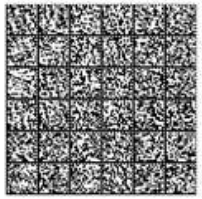
- Pros:
  - Extremely reliable
  - Can be entered by hand if no scanner
- Cons:
  - Low data density: ~3.6K per page best case
  - Common Tools:
    - PowerSploit: 835 kB = 232 pages
    - Mimikatz: 538 kB = 150 pages
  - No exfiltration “compression” advantage

Phase 2

Phase 2



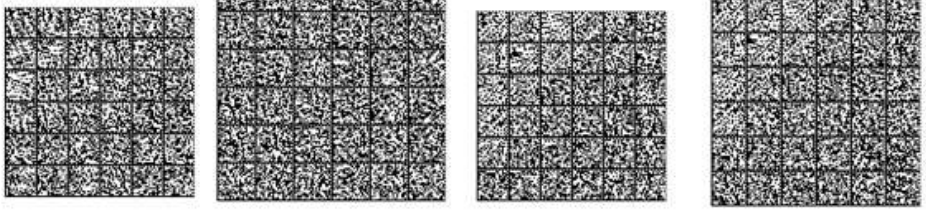




Worked, ~ 49K/page

Fail (online)

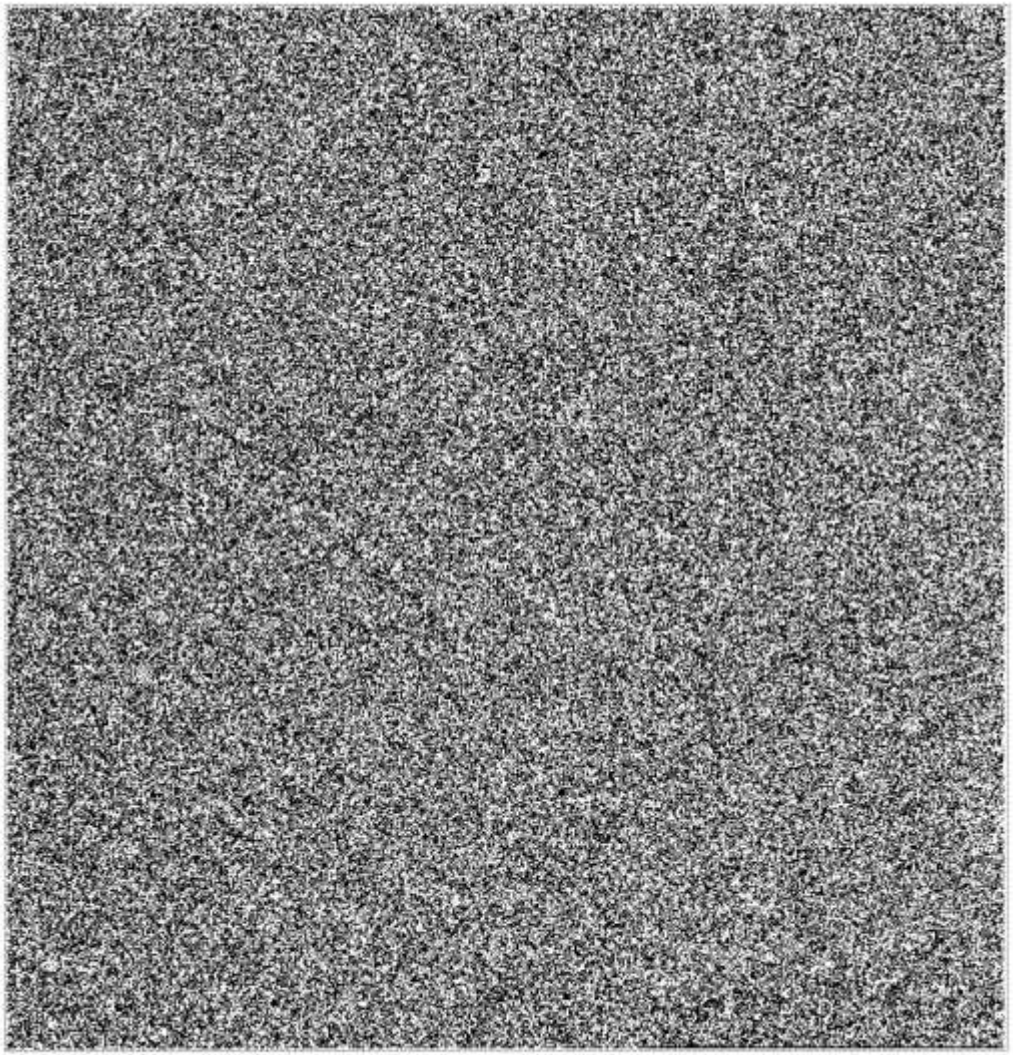
Fail (online)



C40 Base256

Phase 2

- ~ 25K data per page
- 60% error correction
- Good features
  - Timing lines
  - Reed-Solomon FEC
- Different design problem
- I can make it better!



Phase 2 – Big Barcode

- Data grid where each pixel represents one bit state (white = 1, black = 0)

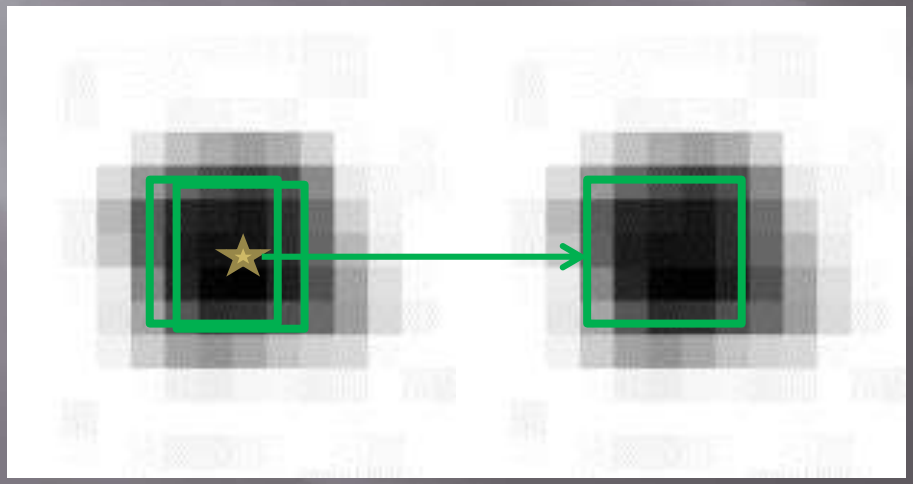
- Printed at 72 dpi, get about 88 bytes across
- ~ 85 kB data per page

Reading a Big Barcode

- Finding the timing marks
  - Start with raster scan across the image

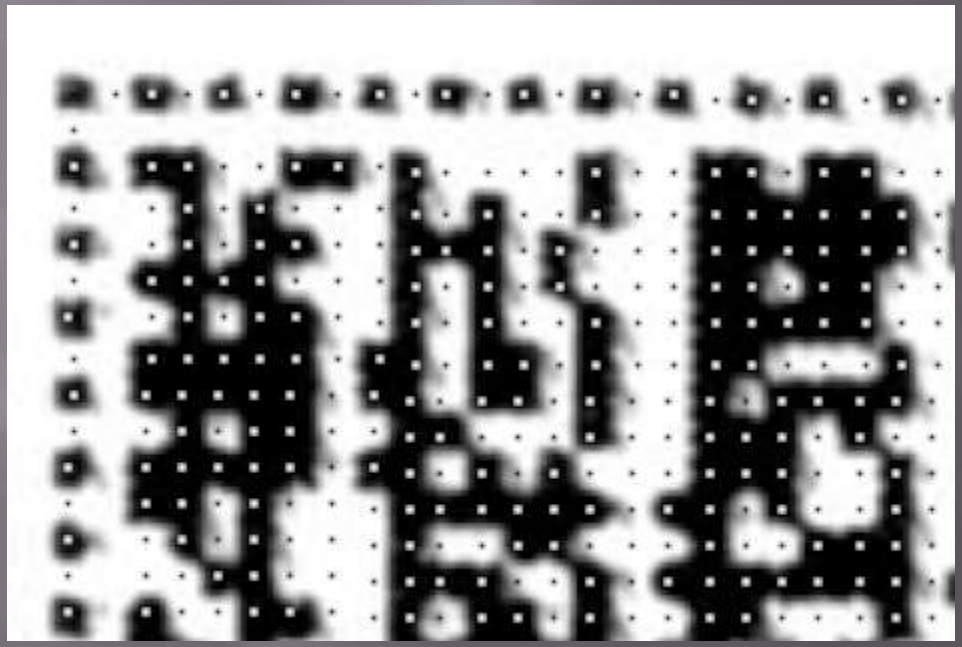
Reading a Big Barcode

- Finding centers of timing marks
- "Wiggle Fit" from "root" pixel
  - Best mask fit



Reading a Big Barcode

- Timing mark location is very successful:
- Once all timing marks found, simply compute a grid of intersections to locate data:



Reading a Big Barcode

- Results:
  - 20K of binary data: 189 bytes missed (0.953% error)
  - 65K of binary data: 491 bytes missed (0.76% error)
  - 72K of ASCII data: 972 bytes missed (1.35% error)

Reed Solomon FEC- Two Types

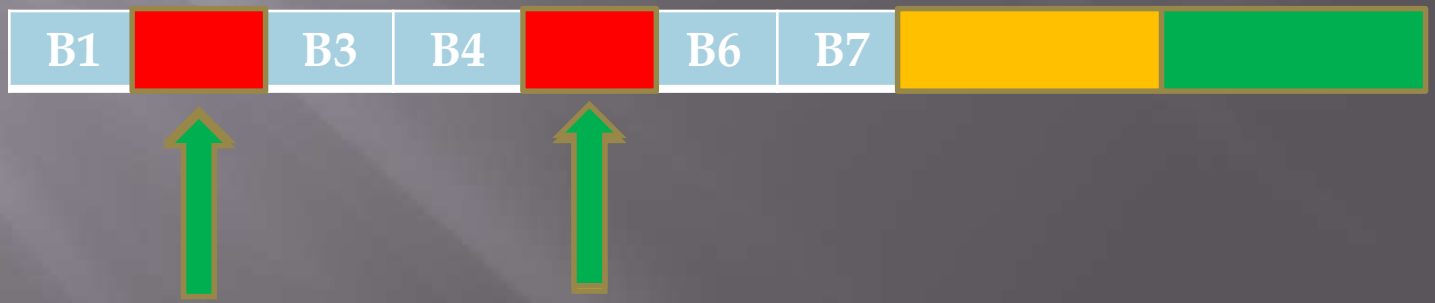
- Forward ERASURE Correction

Block 1
Block 2
Block 3
Block 4

P1	Block 1
P2	Block 2
P3	Block 3
P4	Block 4

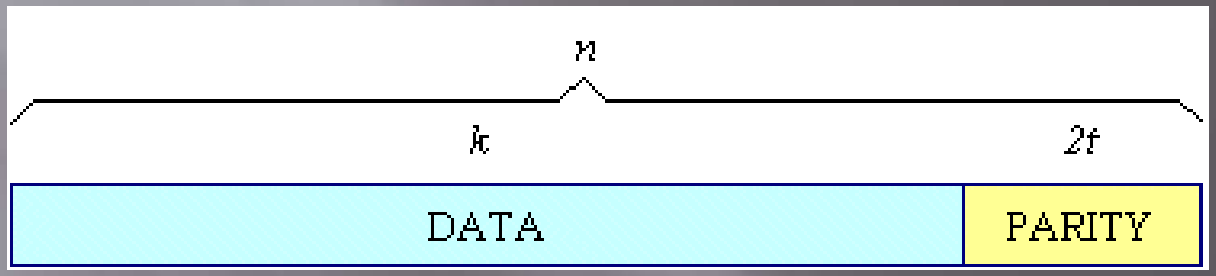
C1	P1	Block 1
C2	P2	Block 2
C3	P3	Block 3
C4	P4	Block 4

- Forward ERROR Correction



Error Correction Details

- Reed Solomon encoding

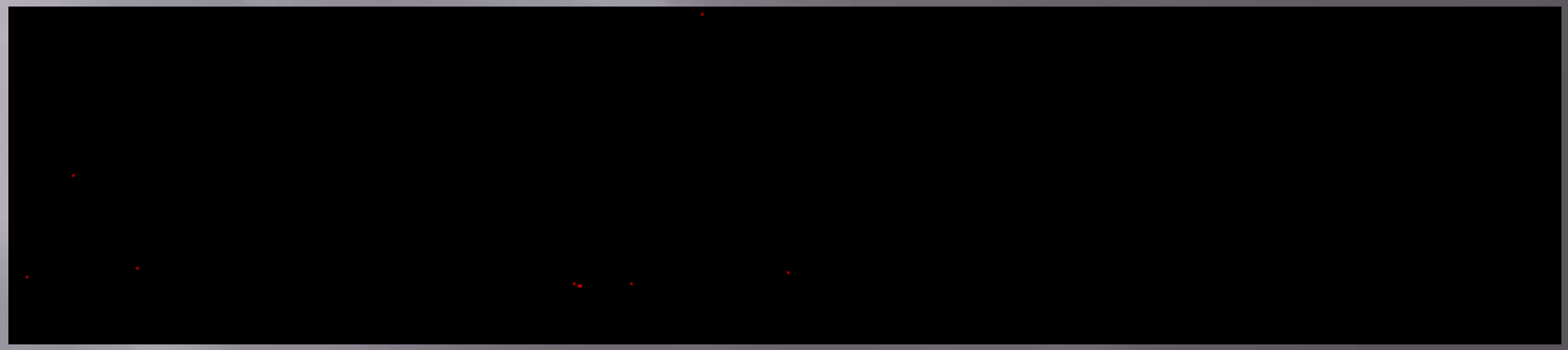


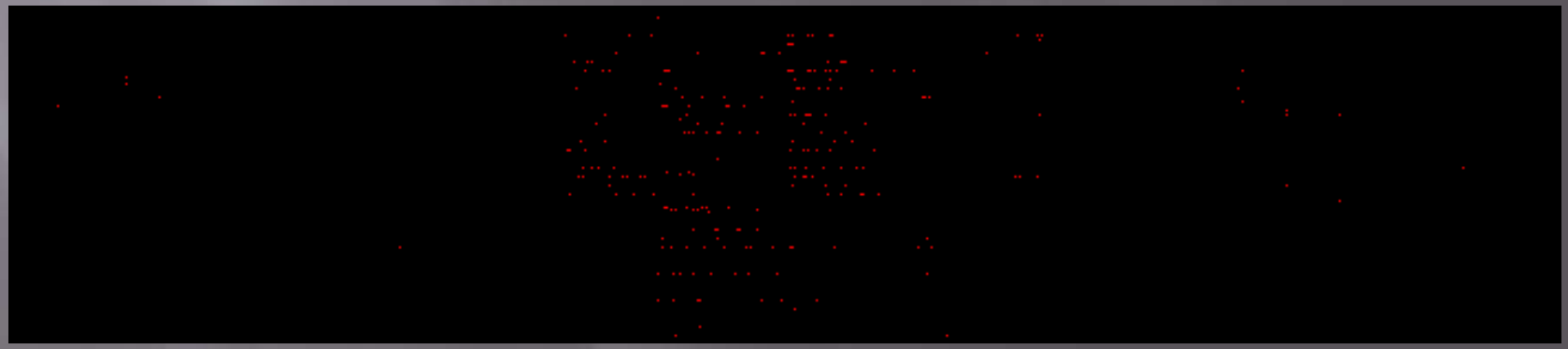
From http://www.cs.cmu.edu/~guyb/realworld/reedsolomon/reed_solomon_codes.html

- Codewords can be 2^s symbols long, each symbol s-bits wide
  - S = 8, codeword is 255 symbols; each symbol 8 bits wide
  - S = 16, codeword is 65535 symbols; each symbol 16 bits wide
  - Codewords can be less than n symbols long
- Can correct up to "t" symbol errors (2 parity symbols required for find and correct each error)

Real-World R-S FEC

- Few open-source error correction libraries
  - Those that do are 2^8 only





Coding an R-S FEC

ReedSolomonTutorial.py – /Users/mrrich/Documents/Programming/RSTut/ReedSolomonTu...

```
    return r

def gf_poly_eval(p,x):
    y = p[0]
    for i in range(1, len(p)):
        y = gf_mul(y,x) ^ p[i]
    return y

def rs_generator_poly(nsym):
    g = [1]
    for i in range(0,nsym):
        g = gf_poly_mul(g, [1, gf_exp[i]])
    return g

def gf_poly_div(dividend, divisor):
    '''Fast polynomial division by using Extended Synthetic Division and opti
    (doesn't work with standard polynomials outside of this galois field, see

    msg_out = bytearray(dividend) # Copy the dividend list and pad with 0 whe
    for i in xrange(len(dividend)-(len(divisor)-1)):
        coef = msg_out[i] # precaching
        if coef != 0: # log(0) is undefined, so we need to avoid that case exp
            for j in xrange(1, len(divisor)): # the divisor is usually monic,
                msg_out[i + j] ^= gf_mul(divisor[j], coef) # equivalent to the
                                                           # (but xoring is fas

    # The resulting msg_out contains both the quotient and the remainder, the
    # (the remainder has necessarily the same degree as the divisor -- not le
    # what we couldn't divide from the dividend), so we compute the index whe
    separator = -(len(divisor)-1)
    return msg_out[:separator], msg_out[separator:] # return quotient, remain

def rs_encode_msg(msg_in, nsym):
    '''Reed-Solomon main encoding function, using polynomial division (algorit
    gen = rs_generator_poly(nsym)
    # Init msg_out with the values inside msg_in and pad with len(gen)-1 bytes
    msg_out = [0] * (len(msg_in) + len(gen)-1)
    # Initializing the Synthetic Division with the dividend (= input message p
    msg_out[:len(msg_in)] = msg_in
```

Ln: 1 Col: 0

ReedSolomon.cpp

RSExp › RSExp › ReedSolomon.cpp › No Selection

```
byte_t ReedSolomon::gf_poly_eval(byteArr_t* p, int x) {
    byte_t y = p->at(0);
    for (int i = 1; i < p->size(); i++) {
        int pati = (*p)[i];
        y = gf_mul(y, x) ^ fixint(pati);
    }
    return fixint(y);
}

void ReedSolomon::gf_poly_div(byteArr_t* dividend, byteArr_t* divisor, byteArr_t* quotient, byteArr_t*
    remainder) {
    // Placeholder
    //int imax = dividend->size() -

}

int ReedSolomon::gf_exp(int i) {
    return fixint(galois_get_ilog_table(w)[fixint(i)]);
}

int ReedSolomon::gf_log(int i) {
    return fixint(galois_get_log_table(w)[fixint(i)]);
}

void ReedSolomon::gf_poly_scale(byteArr_t* p, int x, byteArr_t* result) {
    char* p_char = p->data();
    int numbytes = len_ptr(p);

    // init result and allocate appropriate space
    result->clear();
    result->insert(result->begin(), numbytes, 0);
    char* result_char = result->data();

    if (w == 8) {
        galois_w08_region_multiply(p_char, x, numbytes, result_char, 0);
    } else if (w == 16) {
        galois_w16_region_multiply(p_char, x, numbytes, result_char, 0);
    }

}

void ReedSolomon::gf_poly_add(byteArr_t* p, byteArr_t* q, byteArr_t* result) {
    // First align the polys
```

https://en.wikiversity.org/wiki/Reed%E2%80%93Solomon_codes_for_coders

Big Bar Code Results

- With s=8, k=140 to work reliably
- ~47 kB per page of data (~38 kB of parity)
  - PowerSpoit: 18 pages (vs. 232 pages in hex)
  - Mimikatz: 12 pages (vs. 150 pages in hex)

Demo Time!

Proof Of Concept

- Goal: Using techniques described here, install PowerSploit on a machine

Step	Result
Interpret a page-sized bar code	
Reed-Solomon Encoder/Decoder	
Build Sideload Library	
Encode, Print, Scan, Decode payload with library	
Print, Scan, and load hex encoder/decoder into Excel	
Emplace library using hex OCR method	
Encode/decode using DLL called from Excel	

Future/Branch Research

- Big Bar Code
  - Reduce size of BBC DLL
  - Improve error rates
  - Get 2^16 Reed Solomon FEC working
  - Add color to BBC
- Excel-a-sploit
  - Hex Editor
  - Steganographic encoder/decoder
  - Restore command prompt
  - Direct DLL injection?

Conclusion

- Big Bar Code POC was a success
- Standard office tools provide a lot of power
- If a user can code, a system is not secure
- Innocuous input/output systems can be used for creative purposes

QUESTIONS?