

SXQgaXMgcG9zc2libGUgdG8g  
aW52ZW50IGEgc2luZ2xlIG1h  
Y2hpbmUgd2hpY2ggY2FuIGJl  
IHVzZWQgdG8gY29tcHV0ZSBh  
bnkgY29tcHV0YWJsZSBzZXF1  
ZW5jZS4gSWYgdGhpcyBtYWNo  
aW5lIGF1aW50IGF1aW50IGF1  
d210aG8hIHRhIGF1aW50IGF1  
IGJlZS4gY29tcHV0ZSBzZXF1  
aCBpcyB3cm10dGVuIHROZSBT  
LkQgb2Ygc29tZSBjb21wdXRp  
bmcgbWFjaGluzSBnLzCB0aG  
VuIFUgd21sbCBjb21wdX  
RlIHROZSBzYW11IH  
NlcXV1bmNlIG  
FzIE0uCg  
==

**CDIS**

# CENTER FOR CYBER DEFENCE AND INFORMATION SECURITY



SWEDISH ARMED FORCES



Swedish  
Defence  
University



Swedish  
Defence  
Research  
Agency



National  
Defence Radio  
Establishment



Swedish Civil  
Contingencies  
Agency



# Pegasus Targets

- Journalists
- Activists
- Academics
- Lawyers
- Politicians or Government Officials
- Businessmen
- Religious Figures
- Doctors
- Friends or Relatives
- Prosecutors
- Military officers
- Royals



Jamal Khashoggi



Édith Chabre



Abdelaziz Bouteflika



Emmanuel Macron



Charles Michel



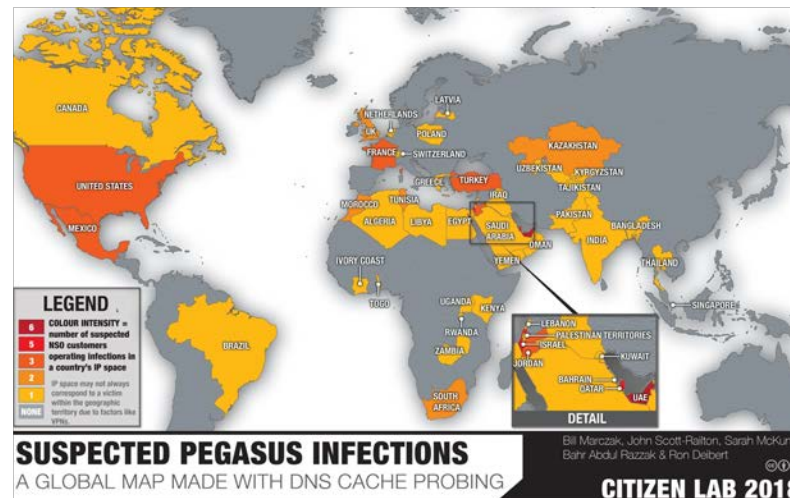
Reddine Bedoui



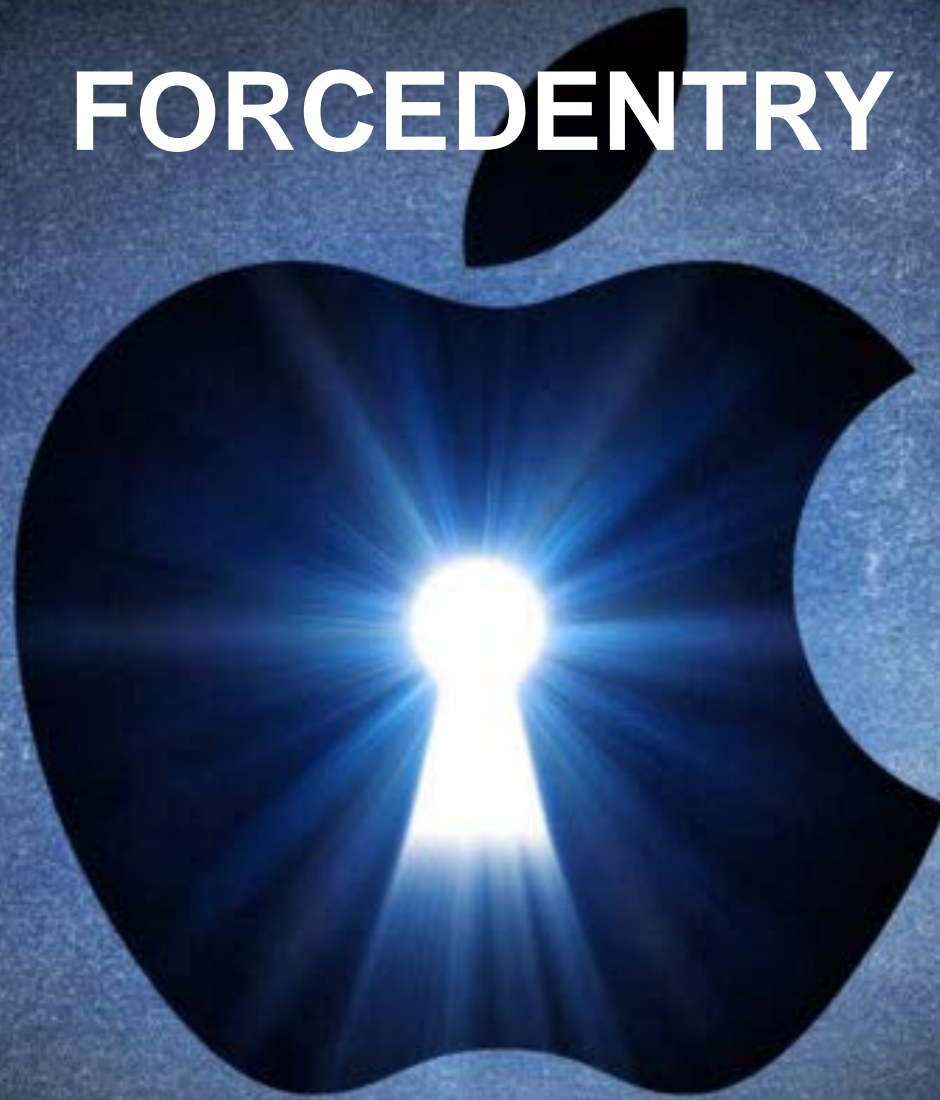
Cyril Ramaphosa



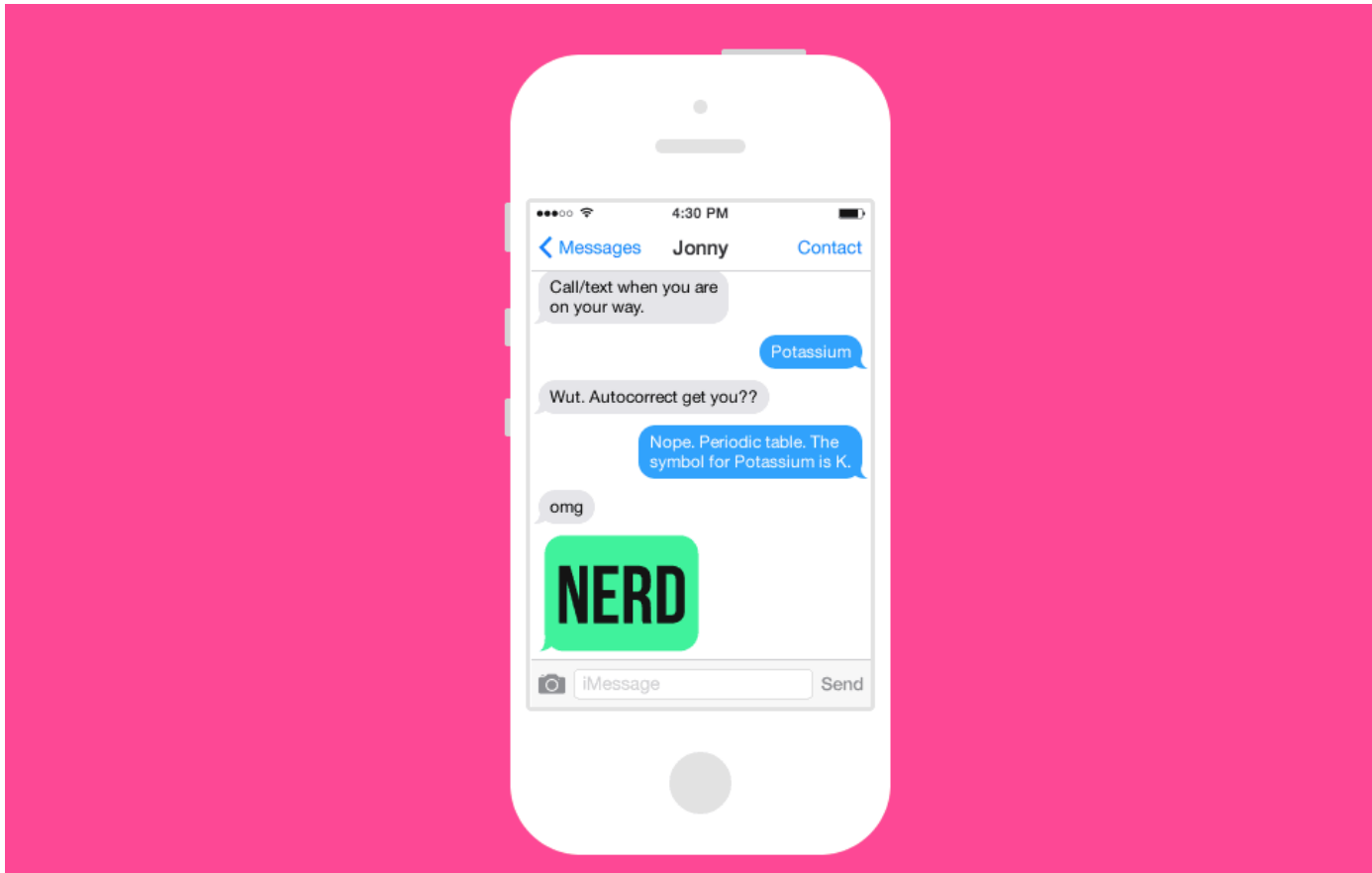
Edouard Philippe



**FORCEDENTRY**

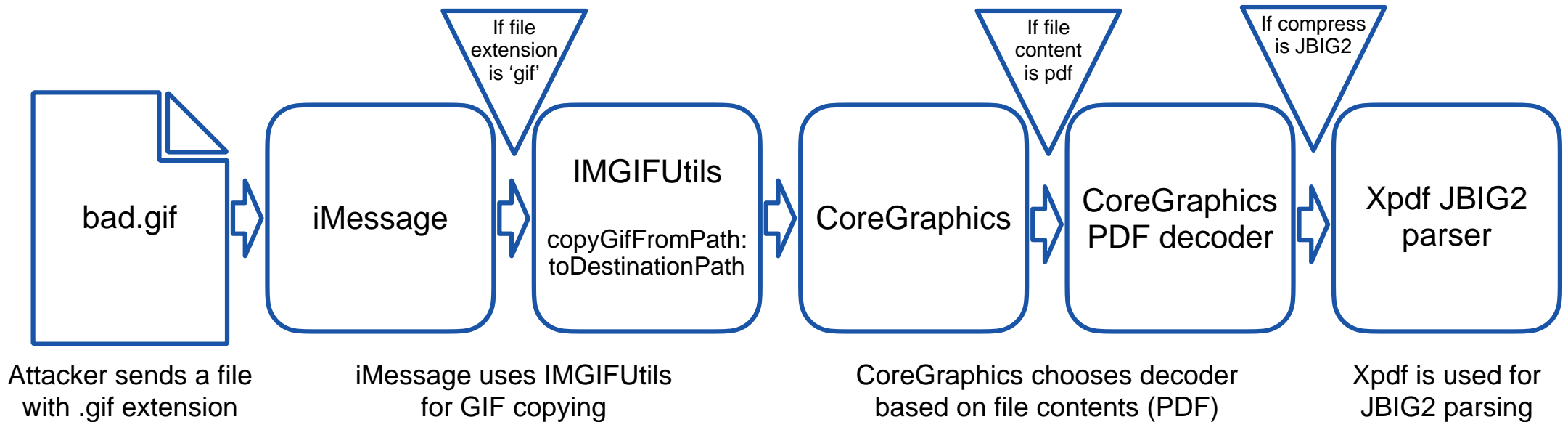


# iPhone iMessage animated GIFs endless loops



5XQqaXKgc09ac01ib0q0d08p  
aW0zK05010ap01uz2x1r0h1  
Y2hptw0q3zhp7agY2Pu1q01  
I8V0w0q0d08p01t0c0V0250h  
h0kgY290c0V00W0z080zX0F1  
Z051284080ad0ap0y0c0V0e0  
0h0100000000000000000000  
d21000000000000000000000  
00h10000000000000000000000  
00h10000000000000000000000  
aC0p0y03c10d00vU080z080T  
I0q027q0c0000000021w000p  
h0c0p0W0j0d100z0N0C00a0  
V0u0P0d110c00j0d1w0x  
011000000000000000000000  
N10000000000000000000000  
F00000000000000000000000  
000000000000000000000000

# Targeting a vulnerability in JBIG2 from a GIF



# Vulnerable code in the Xpdf JBIG2 parser

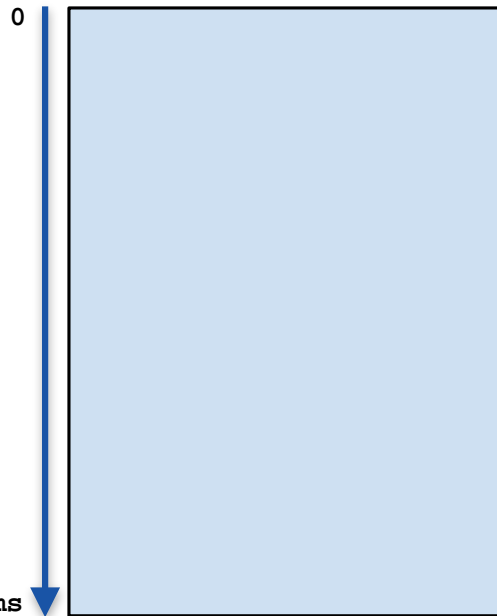
```
    Guint numSyms; // (1)

    numSyms = 0;
    for (i = 0; i < nRefSegs; ++i) {
        if ((seg = findSegment(refSegs[i]))) {
            if (seg->getType() == jbig2SegSymbolDict) {
                numSyms += ((JBIG2SymbolDict *)seg)->getSize(); // (2)
            } else if (seg->getType() == jbig2SegCodeTable) {
                codeTables->append(seg);
            }
        } else {
            error(errSyntaxError, getPos(),
                "Invalid segment reference in JBIG2 text region");
            delete codeTables;
            return;
        }
    }
    ...
    // get the symbol bitmaps
    syms = (JBIG2Bitmap **)gmallocn(numSyms, sizeof(JBIG2Bitmap *)); // (3)

    kk = 0;
    for (i = 0; i < nRefSegs; ++i) {
        if ((seg = findSegment(refSegs[i]))) {
            if (seg->getType() == jbig2SegSymbolDict) {
                symbolDict = (JBIG2SymbolDict *)seg;
                for (k = 0; k < symbolDict->getSize(); ++k) {
                    syms[kk++] = symbolDict->getBitmap(k); // (4)
                }
            }
        }
    }
}
```



# Allocate memory buffer of size numSyms



```
uint numSyms; // (1)
```

```
numSyms = 0;
for (i = 0; i < nRefSegs; ++i) {
    if ((seg = findSegment(refSegs[i]))) {
        if (seg->getType() == jbig2SegSymbolDict) {
            numSyms += ((JBIG2SymbolDict *)seg)->getSize(); // (2)
        } else if (seg->getType() == jbig2SegCodeTable) {
            codeTables->append(seg);
        }
    } else {
        error(errSyntaxError, getPos(),
            "Invalid segment reference in JBIG2 text region");
        delete codeTables;
        return;
    }
}
```

...

```
// get the symbol bitmaps
```

```
syms = (JBIG2Bitmap **)gmallocn(numSyms, sizeof(JBIG2Bitmap *)); // (3)
```

```
kk = 0;
for (i = 0; i < nRefSegs; ++i) {
    if ((seg = findSegment(refSegs[i]))) {
        if (seg->getType() == jbig2SegSymbolDict) {
            symbolDict = (JBIG2SymbolDict *)seg;
            for (k = 0; k < symbolDict->getSize(); ++k) {
                syms[kk++] = symbolDict->getBitmap(k); // (4)
            }
        }
    }
}
```

```
S0GqaXMc09ac01b00p008p
a0S0a000100001000010001
Y2hptm0g02up7ag7p0g01
I0V0a00g010001000000000
h0kg790c0000a0000000000
ZMS10800000000000000000
00100000000000000000000
d2100000000000000000000
00010000000000000000000
a0Bp0y0c010000000000000
I00g020p000000000000000
h0c000F0000000000000000
V0100g01000000000000000
0100000000000000000000
N0c0V10m0100
F000000g
**
```

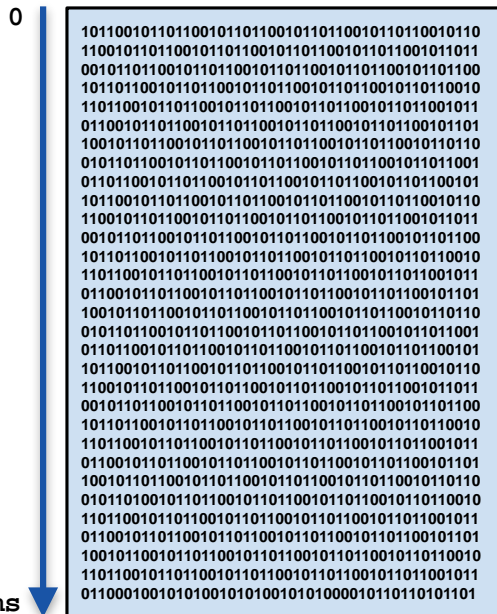


# Write data to memory buffer

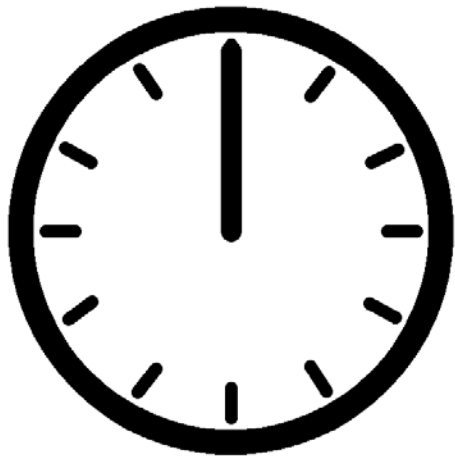
```
uint numSyms; // (1)
```

```
numSyms = 0;
for (i = 0; i < nRefSegs; ++i) {
    if ((seg = findSegment(refSegs[i]))) {
        if (seg->getType() == jbig2SegSymbolDict) {
            numSyms += ((JBIG2SymbolDict *)seg)->getSize(); // (2)
        } else if (seg->getType() == jbig2SegCodeTable) {
            codeTables->append(seg);
        }
    } else {
        error(errSyntaxError, getPos(),
            "Invalid segment reference in JBIG2 text region");
        delete codeTables;
        return;
    }
}
...
// get the symbol bitmaps
syms = (JBIG2Bitmap **)gmallocn(numSyms, sizeof(JBIG2Bitmap *)); // (3)
```

```
kk = 0;
for (i = 0; i < nRefSegs; ++i) {
    if ((seg = findSegment(refSegs[i]))) {
        if (seg->getType() == jbig2SegSymbolDict) {
            symbolDict = (JBIG2SymbolDict *)seg;
            for (k = 0; k < symbolDict->getSize(); ++k) {
                syms[kk++] = symbolDict->getBitmap(k); // (4)
            }
        }
    }
}
}
```



# A sufficiently large object can overflow numSyms



```
    Guint numSyms; // (1)

    numSyms = 0;
    for (i = 0; i < nRefSegs; ++i) {
        if ((seg = findSegment(refSegs[i]))) {
            if (seg->getType() == jbig2SegSymbolDict) {
                numSyms += ((JBIG2SymbolDict *)seg)->getSize(); // (2)
            } else if (seg->getType() == jbig2SegCodeTable) {
                codeTables->append(seg);
            }
        } else {
            error(errSyntaxError, getPos(),
                "Invalid segment reference in JBIG2 text region");
            delete codeTables;
            return;
        }
    }
    ...
    // get the symbol bitmaps
    syms = (JBIG2Bitmap **)gmallocn(numSyms, sizeof(JBIG2Bitmap *)); // (3)

    kk = 0;
    for (i = 0; i < nRefSegs; ++i) {
        if ((seg = findSegment(refSegs[i]))) {
            if (seg->getType() == jbig2SegSymbolDict) {
                symbolDict = (JBIG2SymbolDict *)seg;
                for (k = 0; k < symbolDict->getSize(); ++k) {
                    syms[kk++] = symbolDict->getBitmap(k); // (4)
                }
            }
        }
    }
}
```

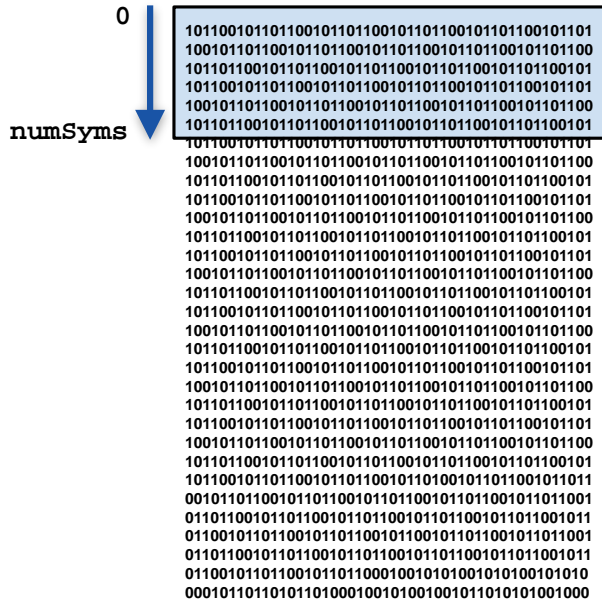
```
S0GqaXMc09e0i1b0q0d08p
a0S0a000i0000i0000i0000i
Y2hptm0q0z0p0g0r0u0g0l
I0V0a0q0d0e0r0t0e0v0S0h
h0k0r090c0000i0a0S0a0X0F1
Z0S0J0B0q080e0d0a0p0y0e0V0l0
000100000000000000000000
d2i0i00000000000000000000
0000000000000000000000000000
a0B0p0y030c010d00v0I0B0s0S0T
I0G0e0T0p0e0000000000000000
h0e0p0W0j0d0i0u0S0N0C0B0a0G
V0u0P0q0d0i0b0c0j0d0i0w0X
010000000000000000000000
N0c0V0l0m0I0G
F0000000g
**
```

# Buffer overflow: big data in a small buffer

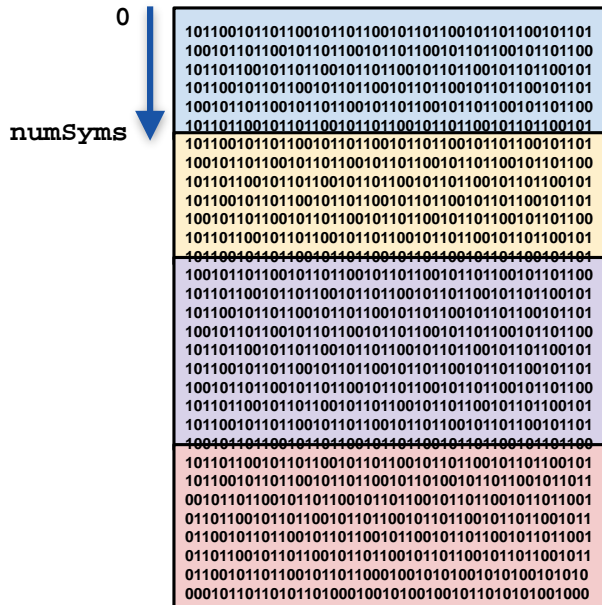
```
uint numSyms; // (1)
```

```
numSyms = 0;
for (i = 0; i < nRefSegs; ++i) {
    if ((seg = findSegment(refSegs[i]))) {
        if (seg->getType() == jbig2SegSymbolDict) {
            numSyms += ((JBIG2SymbolDict *)seg)->getSize(); // (2)
        } else if (seg->getType() == jbig2SegCodeTable) {
            codeTables->append(seg);
        }
    } else {
        error(errSyntaxError, getPos(),
            "Invalid segment reference in JBIG2 text region");
        delete codeTables;
        return;
    }
}
...
// get the symbol bitmaps
syms = (JBIG2Bitmap **)gmallocn(numSyms, sizeof(JBIG2Bitmap *)); // (3)
```

```
kk = 0;
for (i = 0; i < nRefSegs; ++i) {
    if ((seg = findSegment(refSegs[i]))) {
        if (seg->getType() == jbig2SegSymbolDict) {
            symbolDict = (JBIG2SymbolDict *)seg;
            for (k = 0; k < symbolDict->getSize(); ++k) {
                syms[kk++] = symbolDict->getBitmap(k); // (4)
            }
        }
    }
}
}
```



# Buffer overflow overwriting other parts of memory



```
Guint numSyms; // (1)
```

```
numSyms = 0;
for (i = 0; i < nRefSegs; ++i) {
    if ((seg = findSegment(refSegs[i]))) {
        if (seg->getType() == jbig2SegSymbolDict) {
            numSyms += ((JBIG2SymbolDict *)seg)->getSize(); // (2)
        } else if (seg->getType() == jbig2SegCodeTable) {
            codeTables->append(seg);
        }
    } else {
        error(errSyntaxError, getPos(),
            "Invalid segment reference in JBIG2 text region");
        delete codeTables;
        return;
    }
}
```

```
...
// get the symbol bitmaps
syms = (JBIG2Bitmap **)gmallocn(numSyms, sizeof(JBIG2Bitmap *)); // (3)
```

```
kk = 0;
for (i = 0; i < nRefSegs; ++i) {
    if ((seg = findSegment(refSegs[i]))) {
        if (seg->getType() == jbig2SegSymbolDict) {
            symbolDict = (JBIG2SymbolDict *)seg;
            for (k = 0; k < symbolDict->getSize(); ++k) {
                syms[kk++] = symbolDict->getBitmap(k); // (4)
            }
        }
    }
}
```



# The fix is simple: Check for overflow

```
Guint numSyms; // (1)

numSyms = 0;
for (i = 0; i < nRefSegs; ++i) {
    if ((seg = findSegment(refSegs[i]))) {
        if (seg->getType() == jbig2SegSymbolDict) {
            const unsigned int segSize = ((JBIG2SymbolDict *)seg)->getSize();
            if (unlikely(checkedAdd(numSyms, segSize, &numSyms))) {
                error(errSyntaxError, getPos(), "Too many symbols in JBIG2 text region");
                return;
            }
        } else if (seg->getType() == jbig2SegCodeTable) {
            codeTables->append(seg);
        }
    } else {
        error(errSyntaxError, getPos(),
            "Invalid segment reference in JBIG2 text region");
        delete codeTables;
        return;
    }
}
...
// get the symbol bitmaps
syms = (JBIG2Bitmap **)gmallocn(numSyms, sizeof(JBIG2Bitmap *)); // (3)

kk = 0;
for (i = 0; i < nRefSegs; ++i) {
    if ((seg = findSegment(refSegs[i]))) {
        if (seg->getType() == jbig2SegSymbolDict) {
            symbolDict = (JBIG2SymbolDict *)seg;
            for (k = 0; k < symbolDict->getSize(); ++k) {
                syms[kk++] = symbolDict->getBitmap(k); // (4)
            }
        }
    }
}
```





“ There are a thousand  
hacking at the branches  
of evil to one who is  
striking at the root. ”

*~ Henry David Thoreau*

# Derek Noonburg

Xpdf was first released in 1995. It was written, and is still developed, by Derek Noonburg.

- I currently play percussion with the Petaluma Community Band
- I was a grad student in the ECE Department at Carnegie Mellon University in Pittsburgh, PA.
- I was a member of the Kiltie Band, CMU's marching band, and more specifically, the Drumline.



Sandoshin Taiko



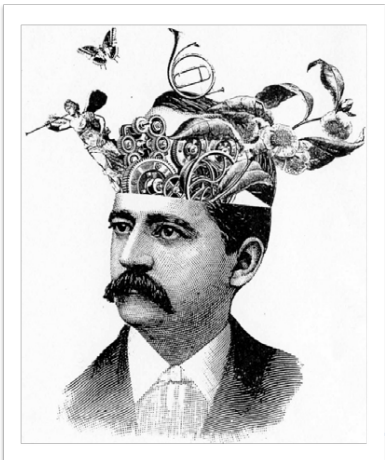
My homemade cider press

**"The Discovery of Bugs in Software: A Case Study"**  
M. Greenberg, S. Malone, **D. Noonburg**, J. Quirke, D. P. Siewiorek  
Keynote address, Intl. Workshop on Computer-Aided Design, Test,  
and Evaluation for Dependability (CADTED), Beijing, China, 1996.



# Derek needs to know many things

- Derek Noonburg needs to understand all about C++, Linux, Windows, iOS, PDFs, the JBIG2 format, ...
- ...but also all about integer overflows...
- ...and all of the other vulnerabilities he might introduce...
- ...and he is not allowed to make mistakes...



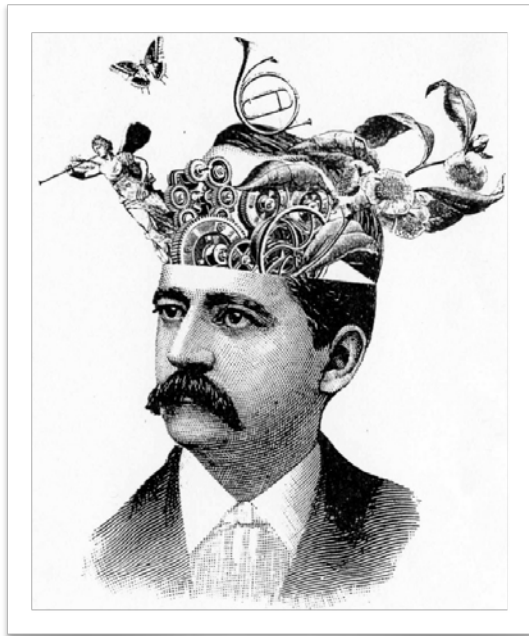
- CERT C++ Secure Coding Section 01 - Preprocessor (PRE) - (869)
- CERT C++ Secure Coding Section 02 - Declarations and Initialization (DCL) - (870)
- CERT C++ Secure Coding Section 03 - Expressions (EXP) - (871)
  - NULL Pointer Dereference - (476)
  - Use of Incorrect Operator - (480)
  - Incorrect Short Circuit Evaluation - (768)
- CERT C++ Secure Coding Section 04 - Integers (INT) - (872)
  - Integer Overflow or Wraparound - (272)
  - Integer Overflow or Wraparound - (272)
  - Integer Overflow or Wraparound - (272)
  - Numeric Truncation Error - (197)
  - Improper Input Validation - (20)
  - Divide By Zero - (369)
  - Return of Pointer Value Outside of Expected Range - (466)
  - Assignment of a Fixed Address to a Pointer - (587)
  - Unchecked Input for Loop Condition - (606)
  - Use of Potentially Dangerous Function - (676)
  - Use of Potentially Dangerous Function - (676)
  - Use of Potentially Dangerous Function - (676)
- CERT C++ Secure Coding Section 05 - Floating Point Arithmetic (FLP) - (873)
  - Divide By Zero - (369)
  - Incorrect Calculation - (682)
  - Function Call With Incorrect Argument Type - (686)
- CERT C++ Secure Coding Section 06 - Arrays and the STL (ARR) - (874)
  - Improper Restriction of Operations within the Bounds of a Memory Buffer - (119)
  - Buffer Overflow or Wraparound - (120)
  - Use of Pointer Subtraction to Determine Size - (469)
  - Improper Initialization - (665)
  - Buffer Access with Incorrect Length Value - (805)
  - C++ String Copying or Initialization with a Null Terminator - (120)
  - Buffer Copy without Checking Size of Input ('Classic Buffer Overflow') - (120)
  - Improper Null Termination - (170)
  - Off-by-one Error - (193)
  - Use of a Broken or Risky Cryptographic Algorithm - (327)
  - Improper Check for Unusual or Exceptional Conditions - (754)
  - Incorrect Type Conversion or Cast - (704)
  - Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection') - (78)
  - Improper Neutralization of Argument Delimiters in a Command ('Argument Injection') - (88)
- CERT C++ Secure Coding Section 08 - Memory Management (MEM) - (876)
  - Improper Restriction of Operations within the Bounds of a Memory Buffer - (119)
  - Wrap-around Error - (128)
  - Incorrect Calculation of Buffer Size - (131)
  - Integer Overflow or Wraparound - (190)
  - Improper Input Validation - (20)
  - Sensitive Information in Resource Not Removed Before Reuse - (226)
  - Improper Clearing of Heap Memory Before Release ('Heap Inspection') - (244)
  - Unchecked Return Value - (252)
  - Unchecked Error Condition - (391)
  - Improper Resource Shutdown or Release - (404)
  - Double Free - (415)
  - Use After Free - (416)
  - NULL Pointer Dereference - (476)
  - Exposure of Core Dump File to an Unauthorized Control Sphere - (528)
  - Free of Memory not on the Heap - (590)
  - Sensitive Data Storage in Improperly Locked Memory - (591)
  - Improper Initialization - (665)
  - Function Call With Incorrectly Specified Argument Value - (687)
  - Unchecked Return Value to NULL Pointer Dereference - (690)
  - Improper Check or Handling of Exceptional Conditions - (703)
  - Improper Check for Unusual or Exceptional Conditions - (754)
  - Mismatched Memory Management Routines - (762)
  - Allocation of Resources Without Limits or Throttling - (770)
  - Untrusted Pointer Dereference - (822)
- CERT C++ Secure Coding Section 09 - Input Output (FIO) - (877)
  - Improper Restriction of Operations within the Bounds of a Memory Buffer - (119)
  - Use of Externally-Controlled Format String - (134)

- Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal') - (22)
- Improper Handling of Unexpected Data Type - (241)
- Incorrect Default Permissions - (276)
- Incorrect Execution-Assigned Permissions - (279)
- Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition') - (362)
  - Time-of-check Time-of-use (TOCTOU) Race Condition - (367)
  - Path Traversal: '/absolute/pathname/here' - (37)
  - Path Traversal: '/absolute/pathname/here' - (37)
  - Path Traversal: '/absolute/pathname/here' - (38)
  - Path Traversal: 'C:\dirname' - (39)
- Unchecked Error Condition - (391)
- Exposure of File Descriptor to Unintended Control Sphere ('File Descriptor Leak') - (403)
- Improper Resource Shutdown or Release - (404)
- Improper Resolution of Path Equivalence - (41)
- Files or Directories Accessible to External Parties - (552)
- Improper Link Resolution Before File Access ('Link Following') - (59)
  - UNIX Hard Link - (62)
  - Windows Symbolic Link Following (.LNK) - (64)
  - Windows Symbolic Link Following (.LNK) - (64)
- Improper Handling of Windows Device Names - (67)
- File Operations on Resources in Single-Operation Context - (675)
- File Operations on Resources in Single-Operation Context - (675)
- External Control of File Name or Path - (73)
- Incorrect Permission Assignment for Critical Resource - (732)
- Allocation of Resources Without Limits or Throttling - (770)
- CERT C++ Secure Coding Section 10 - Environment (ENV) - (878)
  - Improper Restriction of Operations within the Bounds of a Memory Buffer - (119)
  - Untrusted Search Path - (426)
  - Duplicate Key in Associative List (Alist) - (462)
  - Incorrect Control Flow Scoping - (705)
  - Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection') - (78)
  - Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection') - (78)
  - Improper Neutralization of Argument Delimiters in a Command ('Argument Injection') - (88)
- CERT C++ Secure Coding Section 11 - Signals (SIG) - (879)
  - Signal Handler Use of a Non-reentrant Function - (479)
  - Improper Synchronization - (662)
- CERT C++ Secure Coding Section 12 - Exceptions and Error Handling (ERR) - (880)
  - Generator of Error Message Containing Sensitive Information - (209)
  - Detection of Error Condition Without Action - (390)
  - Unchecked Error Condition - (391)
  - Improper Cleanup on Thrown Exception - (460)
  - Exposure of Sensitive System Information to an Unauthorized Control Sphere - (497)
  - Missing Standardized Error Handling Mechanism - (544)
  - Improper Check or Handling of Exceptional Conditions - (703)
  - Incorrect Control Flow Scoping - (705)
  - Improper Check for Unusual or Exceptional Conditions - (754)
  - Improper Handling of Exceptional Conditions - (755)
- CERT C++ Secure Coding Section 13 - Object Oriented Programming (OOP) - (881)
- CERT C++ Secure Coding Section 14 - Concurrency (CON) - (882)
  - Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition') - (362)
    - Race Condition within a Thread - (366)
    - Improper Resource Shutdown or Release - (404)
    - Exposure of Data Element to Wrong Session - (488)
    - Missing Release of Resource after Effective Lifetime - (772)
- CERT C++ Secure Coding Section 49 - Miscellaneous (MSC) - (883)
  - Improper Encoding or Escaping of Output - (116)
  - Compiler Removal of Code to Clear Buffers - (14)
  - Improper Handling of Unicode Encoding - (176)
  - Improper Input Validation - (20)
  - Use of a Broken or Risky Cryptographic Algorithm - (327)
  - Use of Insufficiently Random Values - (330)
  - Use of Incorrect Operator - (480)
  - Comparing instead of Assigning - (482)
  - Dead Code - (561)
  - Assignment to Variable without Use - (563)
  - Expression is Always False - (570)
  - Expression is Always True - (571)
  - Incorrect Comparison - (697)
  - Incorrect Type Conversion or Cast - (704)



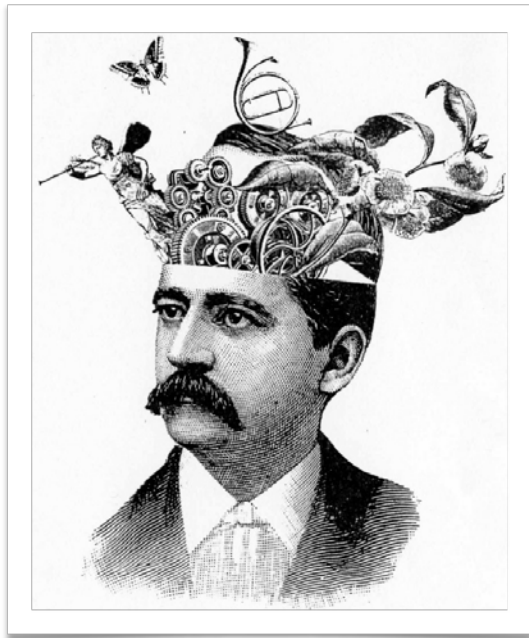


# Root cause: Cognitive complexity

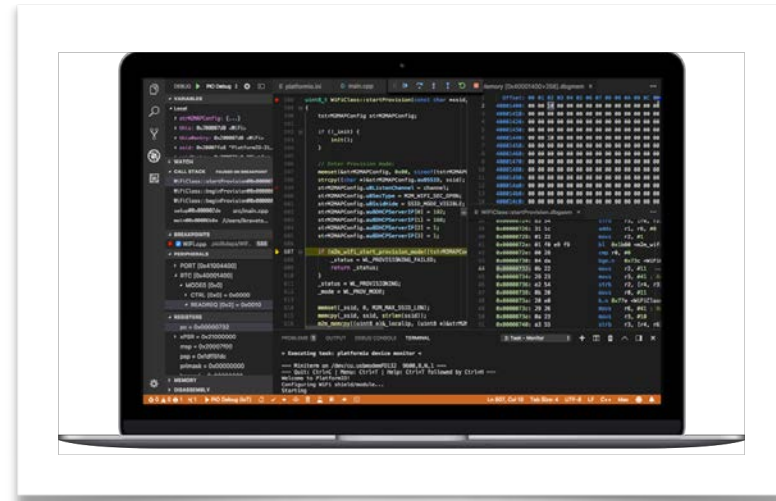


Cognitive complexity

# Developers need cognitive assistance



Cognitive complexity

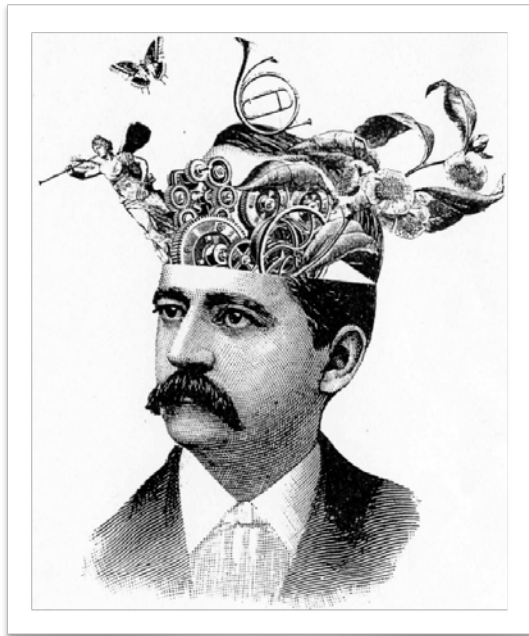


Better tools and methods for secure software development

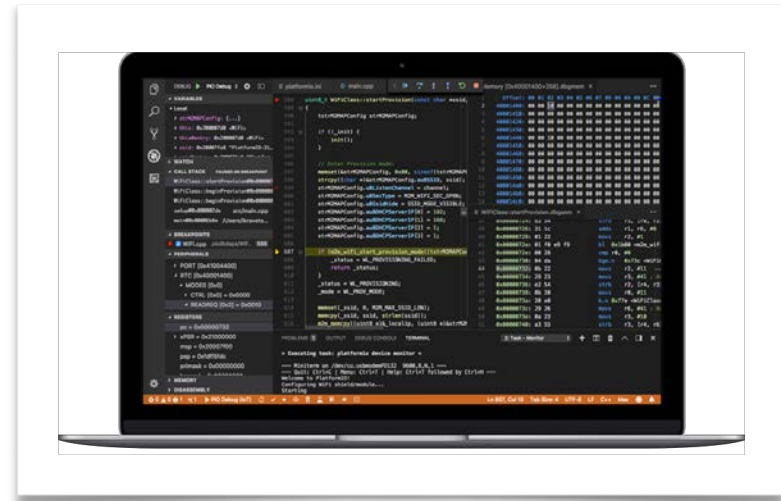
- New programming languages
- New operating systems
- New integrated development environments
- Static-analysis tools
- Dynamic testing tools
- Verification tools
- ...



# Root cause: Cognitive complexity



Cognitive complexity



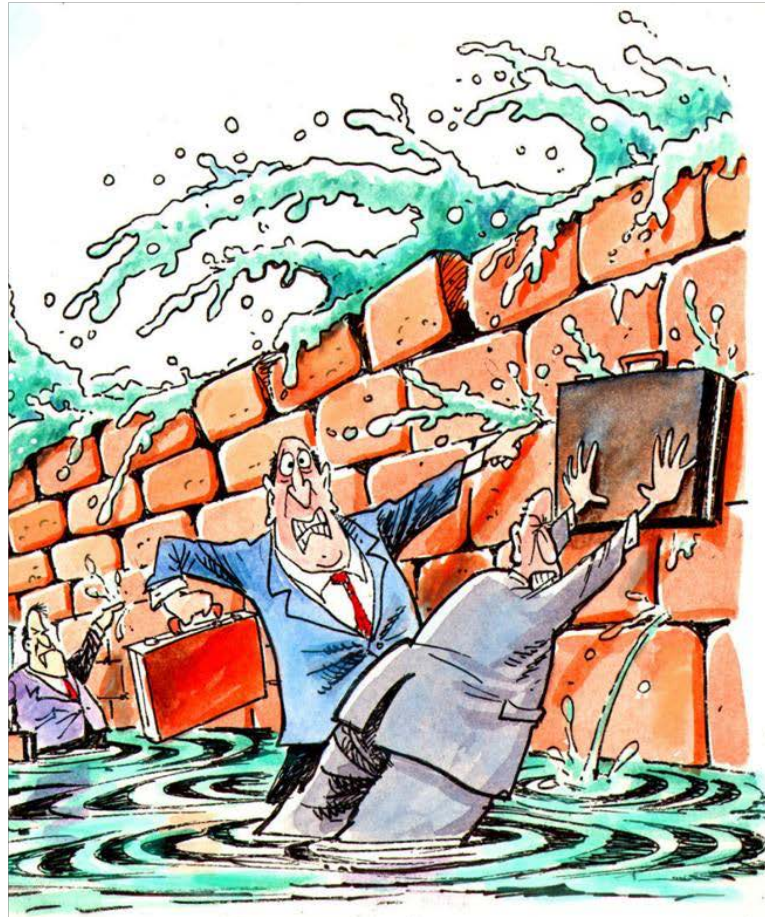
Better tools for secure software development



Research and innovation

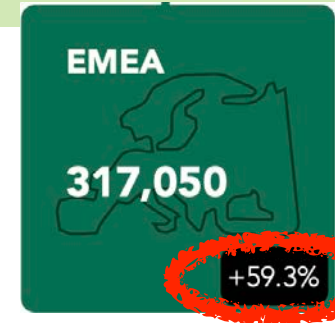
```
5WQqaXKgc09ac01ibDtpd08g  
aW5ZKX501I0ccr1uz2x1r0h1  
YzhpTm9jd2hpY2gY2pUa0l  
I0V0cW9qd09r1tccVU5Zm5h  
hskgY293c0V0VWwzS8Bz2XF1  
Zm5j284g8VedUpoyeVWNo  
c011  
d211v10001000100010001  
202111  
aC8pY83c1000vU180z88T  
I0g027ye10c101021w8Xp  
hmc9wPj01010SNTCB0a0  
Vu1P0gd10001021w4x  
N110R-c0m1W110  
N1cV10m110  
P1000g  
**
```

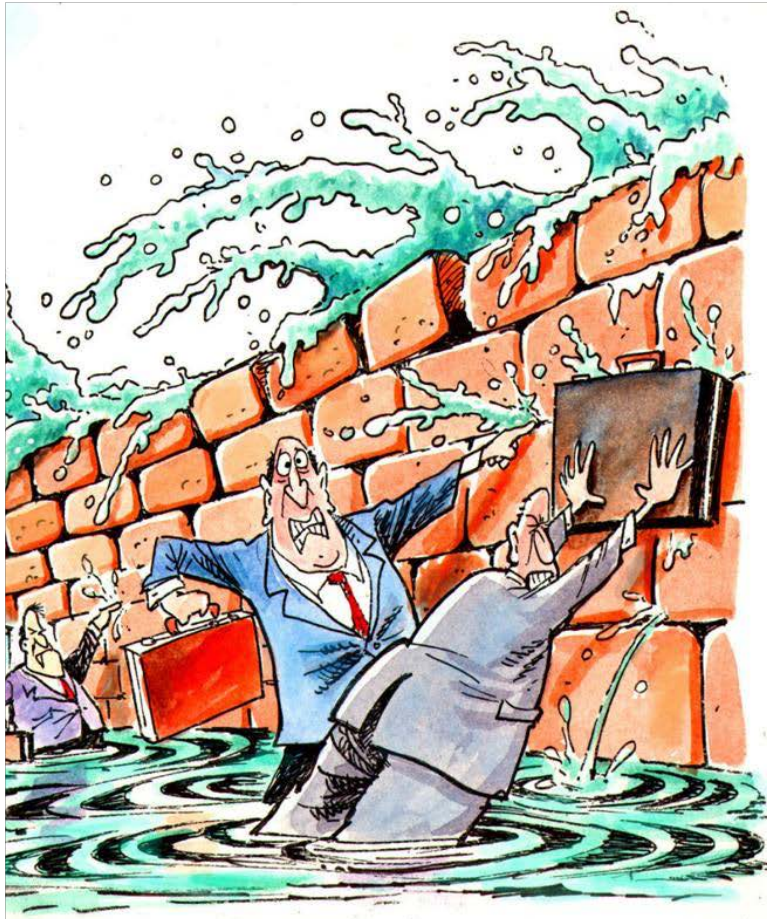
# Insecure systems require many defenders



Despite adding more than 464,000 workers in the past year, the cybersecurity workforce gap has grown more than twice as much as the workforce.

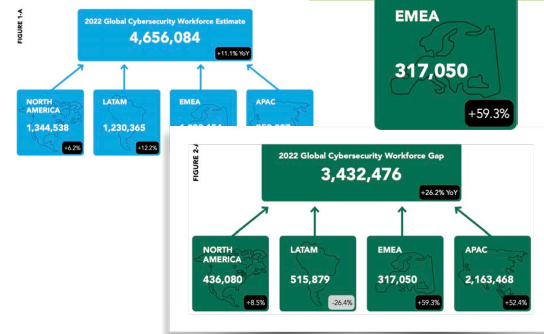
## The Cybersecurity Workforce Gap 2022



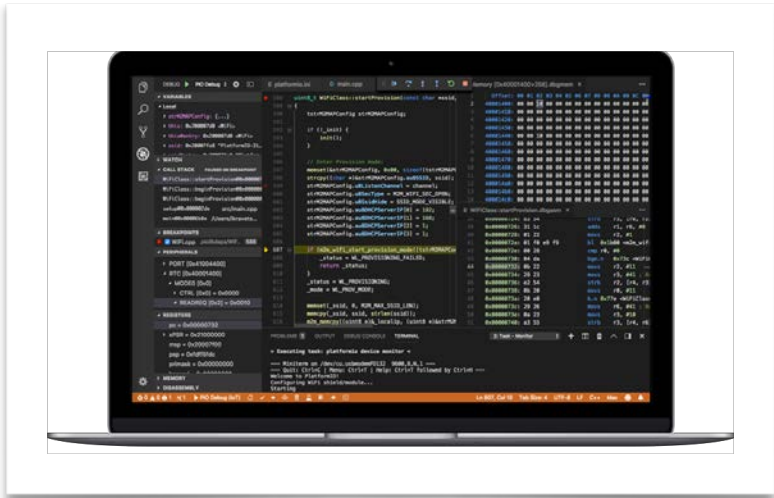


## The Cybersecurity Workforce Gap 2022

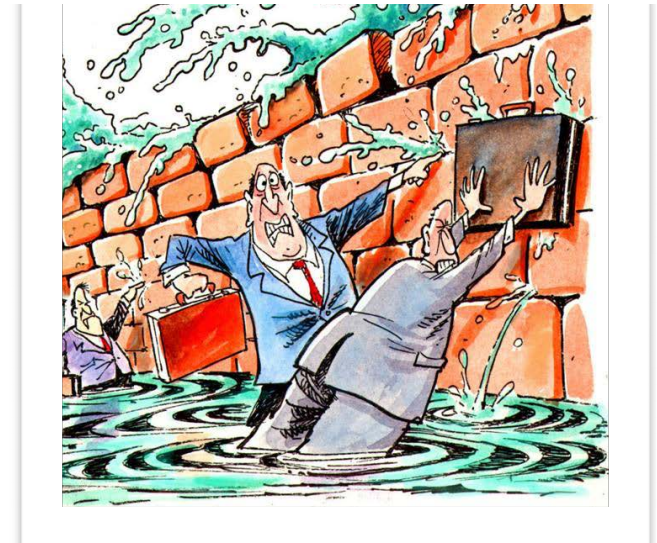
Despite adding more than 464,000 workers in the past year, the cybersecurity workforce gap has grown more than twice as much as the workforce.



Education



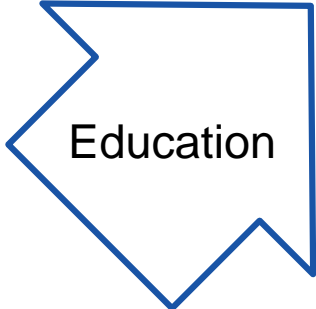
Better tools and methods



More defenders



Research  
Innovation



Education

SXQgaXMgcG9zc2libGUgdG8g  
aW52ZW50IGEgc2luZ2xlIG1h  
Y2hpbmUgd2hpY2ggY2FuIGJl  
IHVzZWQgdG8gY29tcHV0ZSBh  
bnkgY29tcHV0YWJsZSBzZXFl  
ZW5jZS4gSWYgdGhpcyBtYWNo  
aW5lIGF1d283W0pZWQg  
d2l0aBhIRhGgdGhl  
IGJlZ2l0eWlywSaGlj  
aCBpcyB3cm10dGVuIHROZSBT  
LkQgb2Ygc29tZSBjb21wdXRp  
bmcgbWFjaGluzSBnLlCB0aG  
VuIFUgd2lscCBjb21wdX  
RlIHROZSBzYW1lIH  
NlcXVlbnNlIG  
FzIE0uCG  
==

**CENTER FOR CYBER DEFENCE  
AND INFORMATION SECURITY**

SXQgaXMgcG9zc2libGUgdG8g  
aW52ZW50IGEgc2luZ2xlIG1h  
Y2hpbmUgd2hpY2ggY2FuIGJl  
IHVzZWQgdG8gY29tcHV0ZSBh  
bnkgY29tcHV0YWJsZSBzZXFl  
ZW5jZS4gSWYgdGhpcyBtYWNo  
aW5lIGF1d283W0pZWQg  
d2l0aBhIRhGgdGhl  
IGJlZ2l0eWlywSaGlj  
aCBpcyB3cm10dGVuIHROZSBT  
LkQgb2Ygc29tZSBjb21wdXRp  
bmcgbWFjaGluzSBnLlCB0aG  
VuIFUgd2lscCBjb21wdX  
RlIHROZSBzYW1lIH  
NlcXVlbnNlIG  
FzIE0uCG  
==