CDIS

CENTER FOR CYBER DEFENCE AND INFORMATION SECURITY

KTH VETENSKAP OCH KONST

SWEDISH ARMED FORCES

Swedish Defence University

Swedish Defence Research Agency

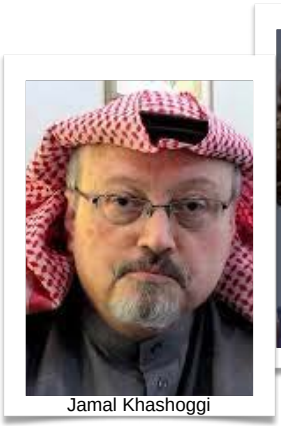
National Defence Radio Establishment

MSB Swedish Civil Contingencies Agency

PEGASUS

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

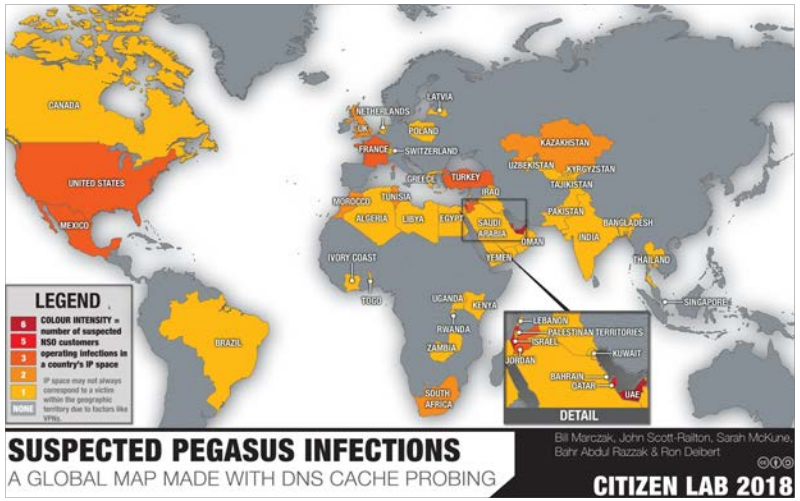

Emmanuel Macron


Charles Michel


reddine Bedoui

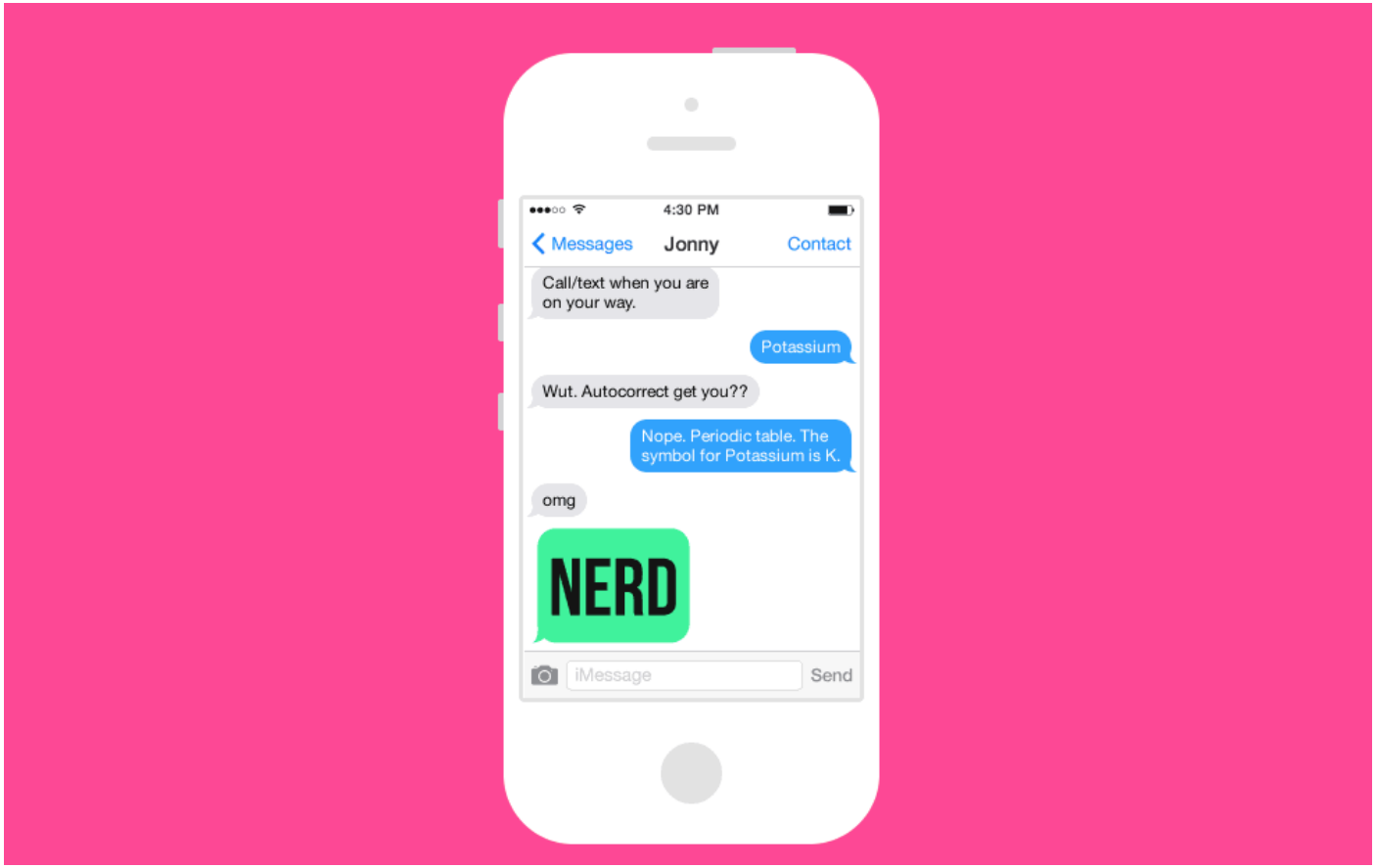

Cyril Ramaphosa


Édouard Philippe

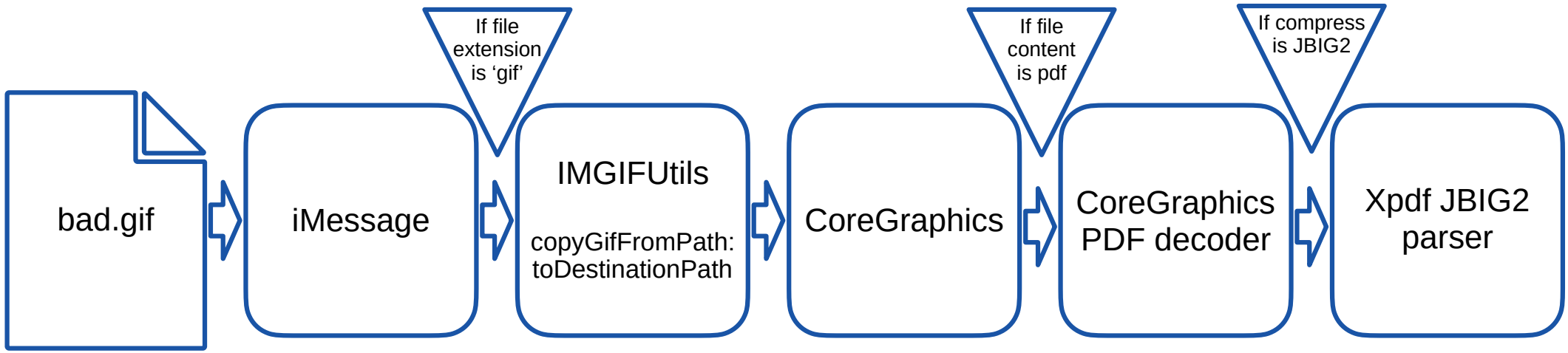



FORCEDENTRY

iPhone iMessage animated GIFs endless loops



Targeting a vulnerability in JBIG2 from a GIF



Attacker sends a file with .gif extension

iMessage uses IMGIFUtils for GIF copying

CoreGraphics chooses decoder based on file contents (PDF)

Xpdf is used for JBIG2 parsing

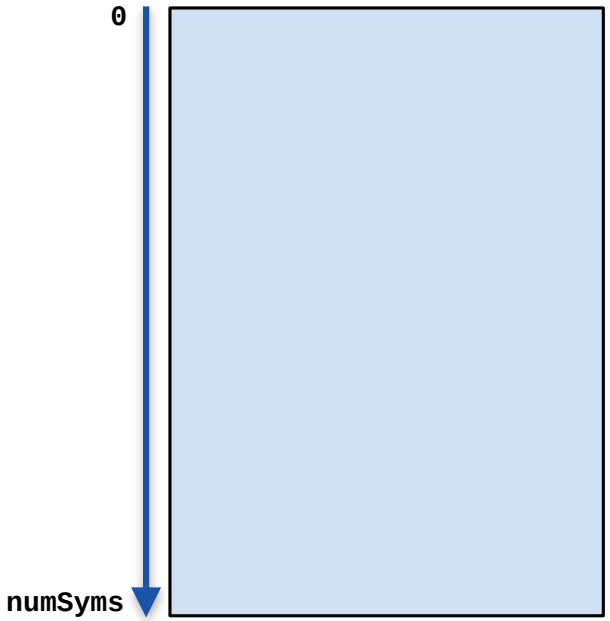
Vulnerable code in the Xpdf JBIG2 parser

```
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

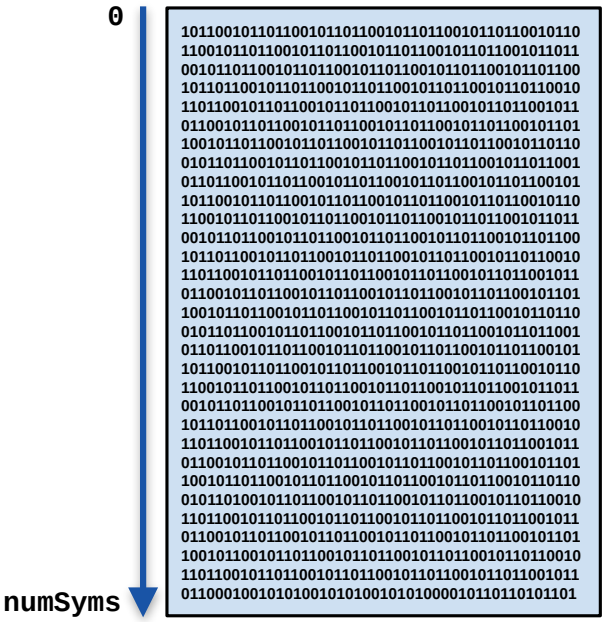



```
Guint numSyms; // (1)

  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)
        }
      }
    }
  }
```

Write data to memory buffer

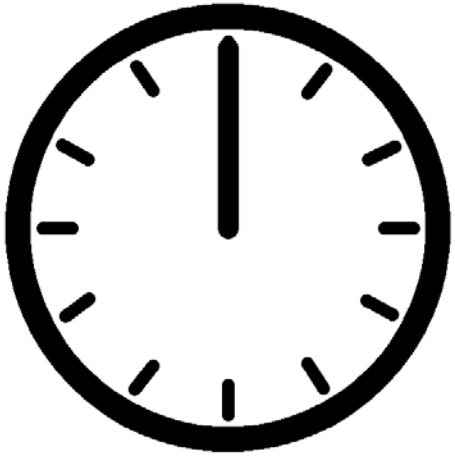



```
Guint numSyms; // (1)

  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)
        }
      }
    }
  }
```

A sufficiently large object can overflow numSyms

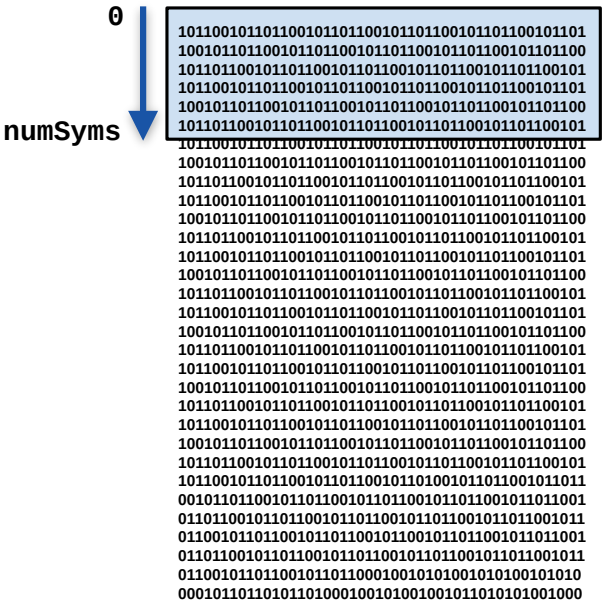


```
Guint numSyms; // (1)

  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)
        }
      }
    }
  }
```

Buffer overflow: big data in a small buffer

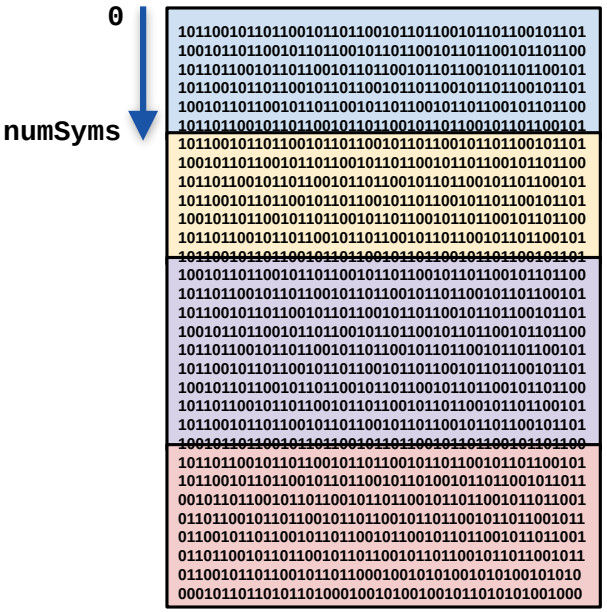



```
Guint numSyms; // (1)

  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)
        }
      }
    }
  }
```

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.

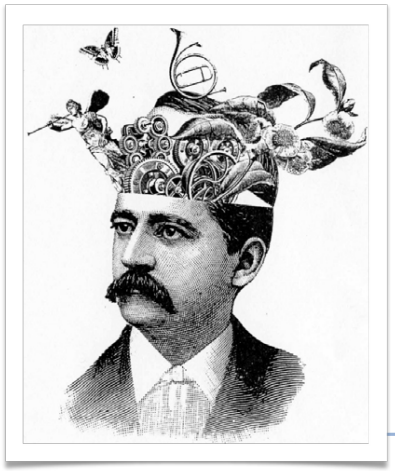
Sandoshin Taiko

My homemade cider press

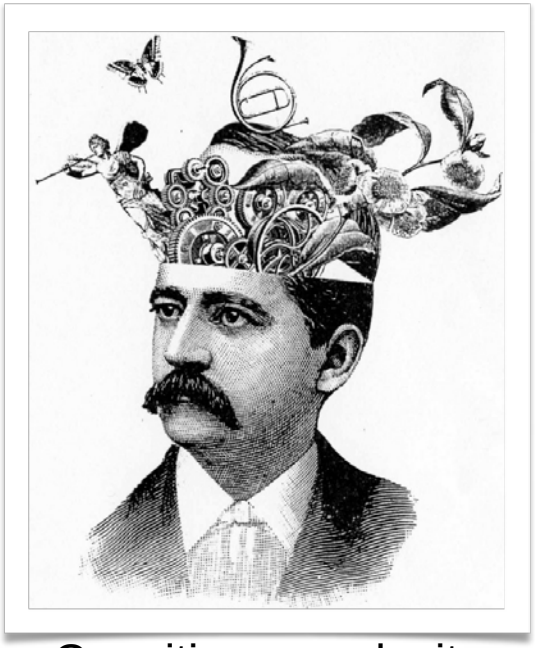
"The Discovery of Bugs in Software: A Case Study"
M. Greenberg, S. Malone, **D. Noonburg**, J. Quirke, D. P. Siewiorek
Keynote address, Intl. Workshop on Computer-Aided Design, Test, and Evaluation for Dependability (CADTED), Beijing, China, 1996.

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…

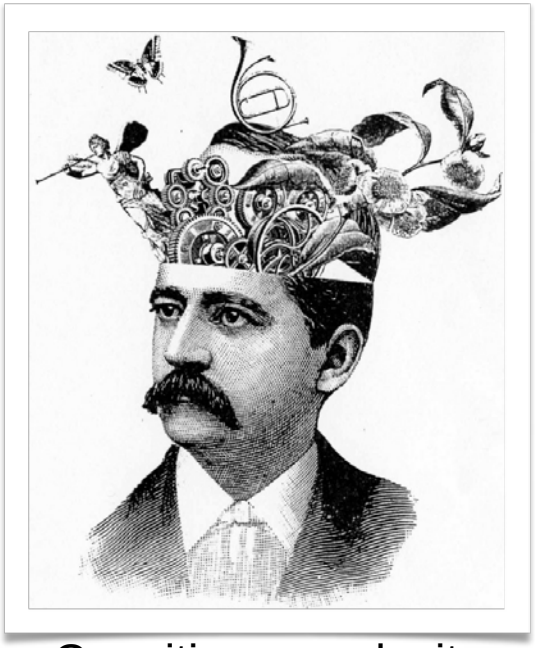


Root cause: Cognitive complexity

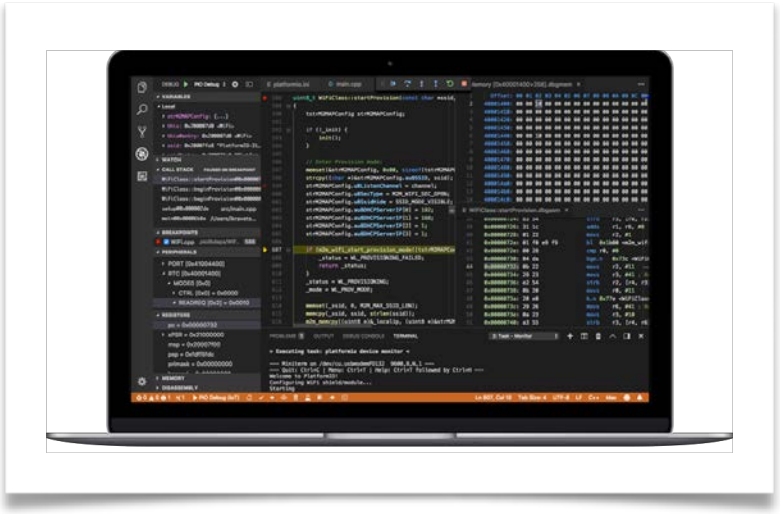


Cognitive complexity

Developers need cognitive assistance



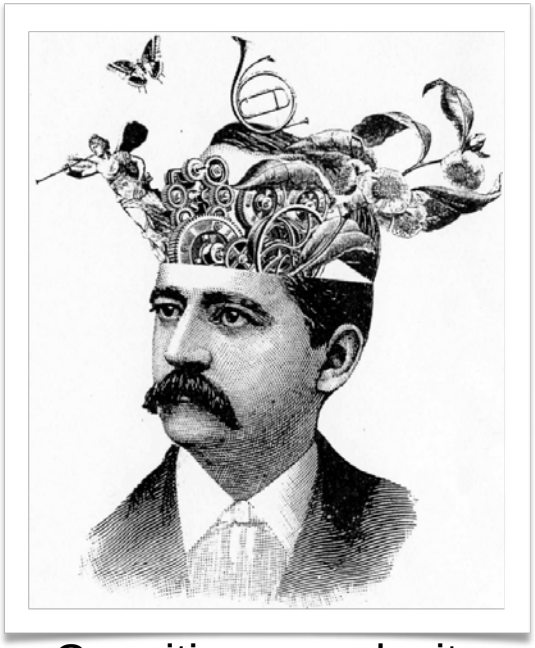
Cognitive complexity



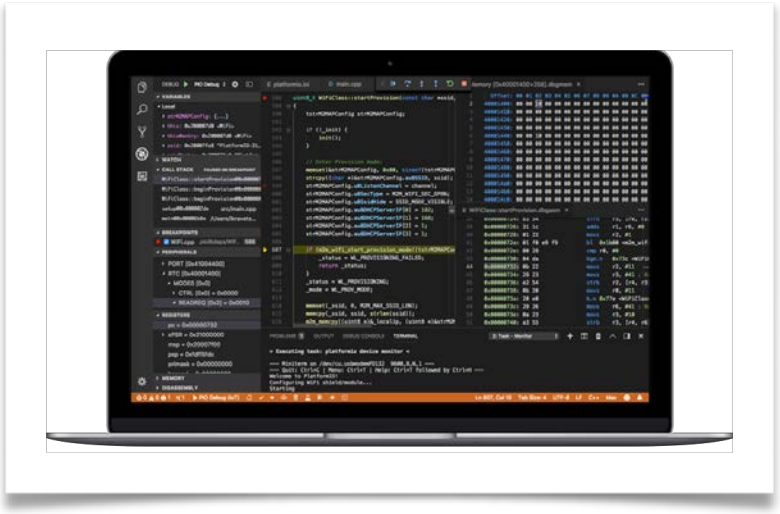
Better tools and methods for secure software development

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

Root cause: Cognitive complexity



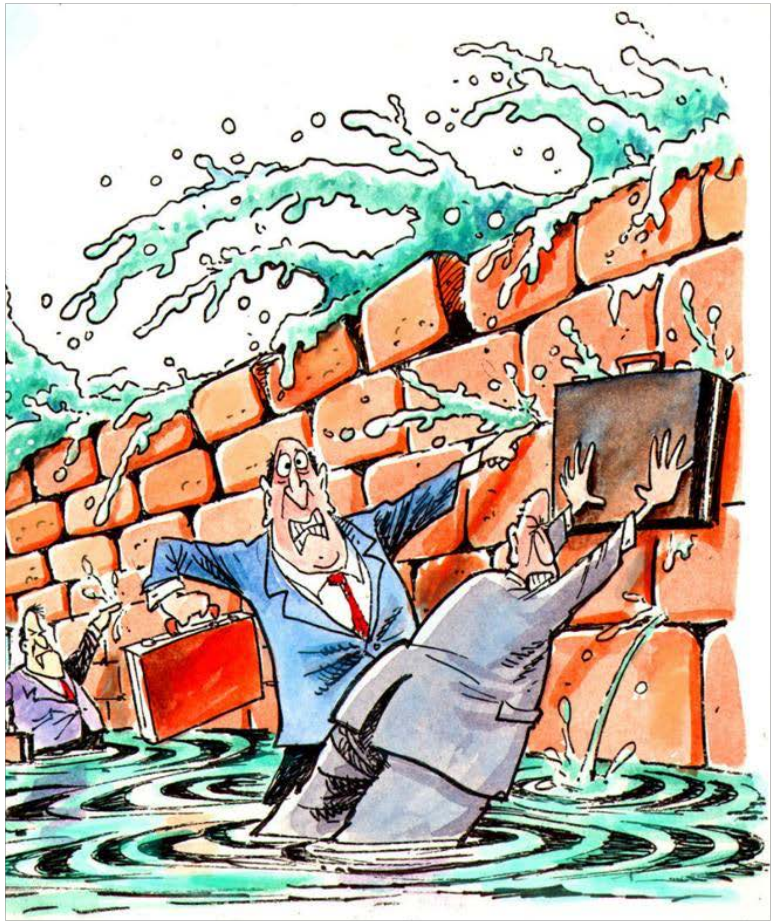
Cognitive complexity



Better tools for secure software development

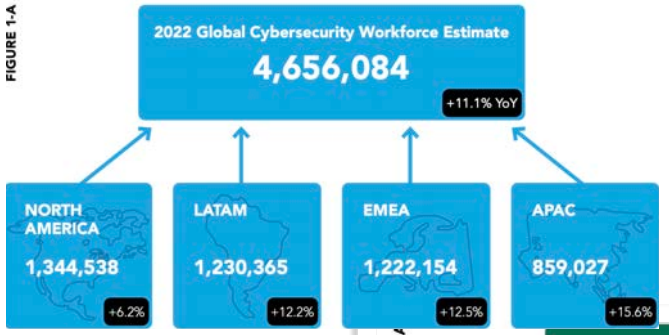
Research and innovation

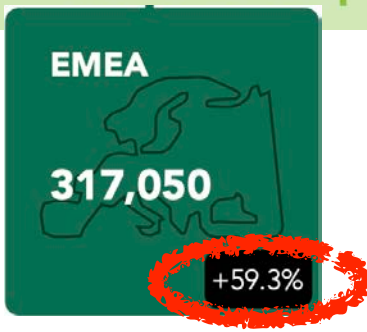
Insecure systems require many defenders

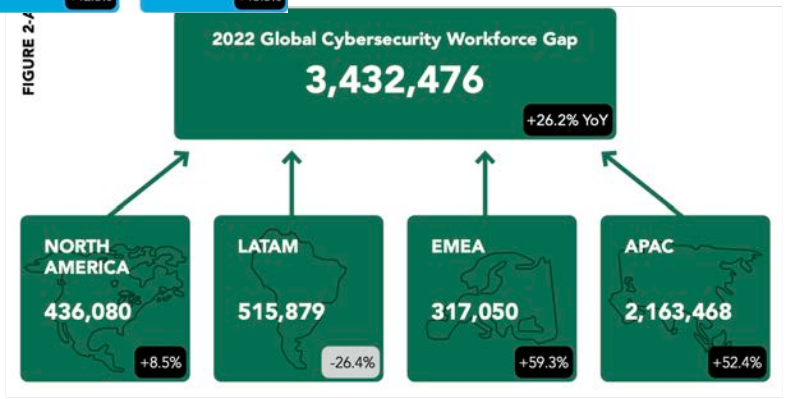


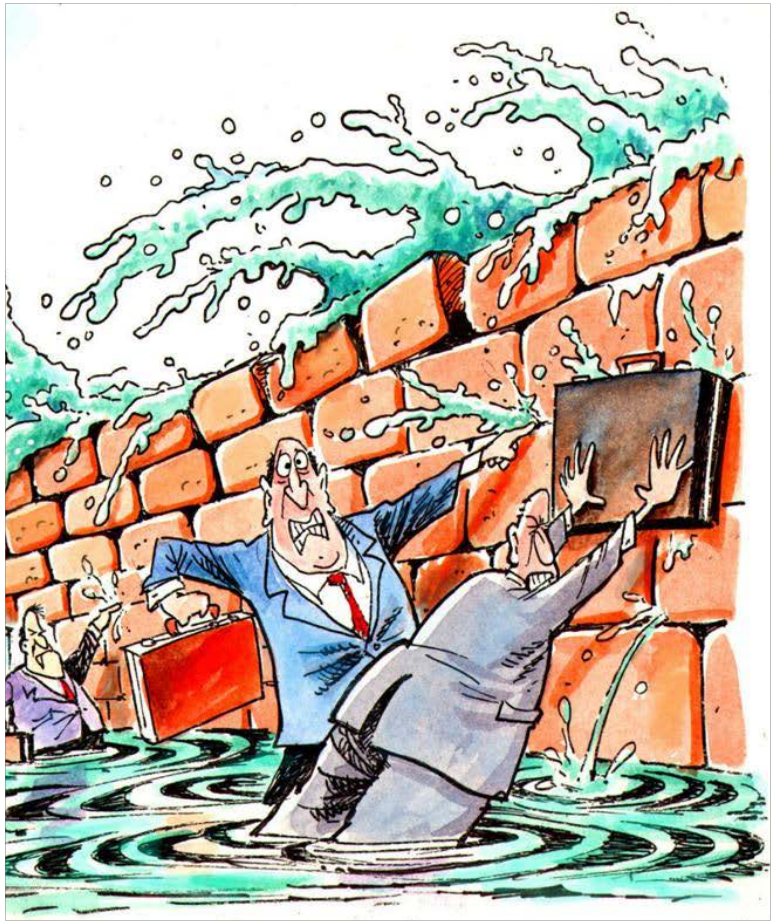
The Cybersecurity Workforce Gap 2022

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.



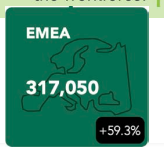


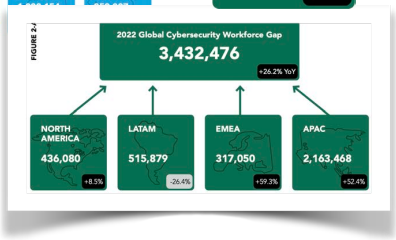




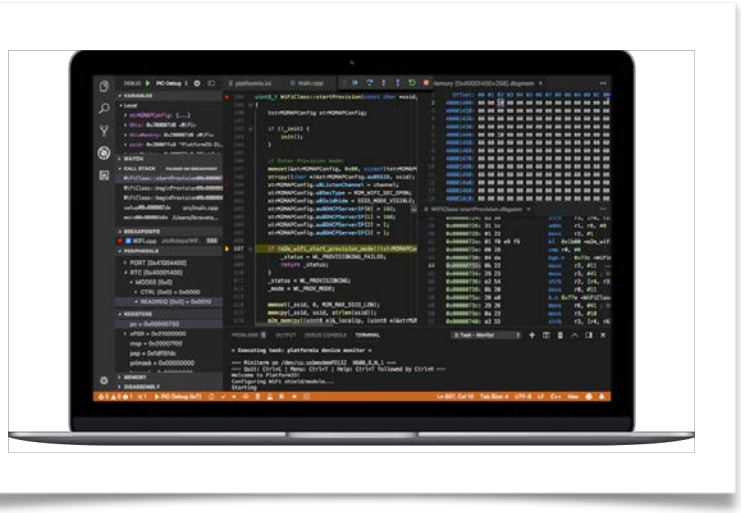
The Cybersecurity Workforce Gap 2022

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.

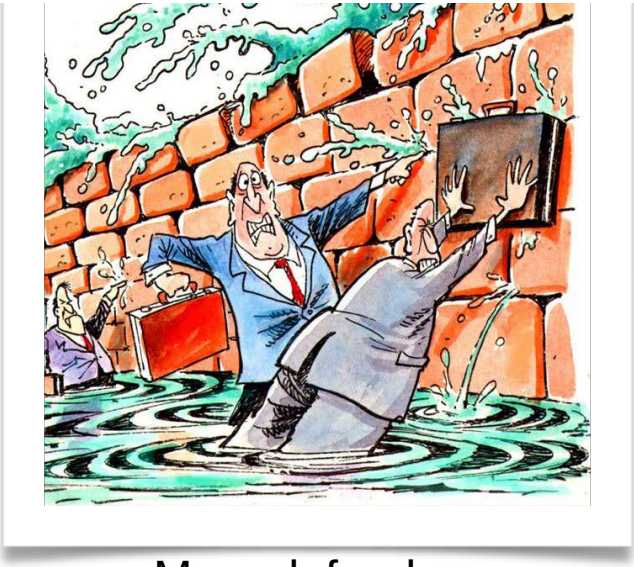




Education



Better tools and methods



More defenders

Research
Innovation

Education

SXQgaXMgcG9zc2libGUgdG8g
aW52ZW50IGEgc2luZ2xlIG1h
Y2hpbmUgd2hpY2ggY2FuIGJl
IHVzZWQgdG8gY29tcHV0ZSBh
bnkgY29tcHV0YWJsZSBzZXF1
ZW5jZS4gSWYgdGhpcyBtYWNo
aW5lIC[illegible]pZWQg
d2l0a[illegible]gdGhl
IGJlZ[illegible]aGlj
aCBpcyB3cml0dGVuIHRoZSBT
LkQgb2Ygc29tZSBjb21wdXRp
bmcgbWFjaGluZSBNLCB0aG
VuIFUgd2lsbCBjb21wdX
RlIHRoZSBzYW1lIH
NlcXVlbmNlIG
FzIE0uCg
==

CDIS

CENTER FOR CYBER DEFENCE
AND INFORMATION SECURITY

SXQgaXMgcG9zc2libGUgdG8g
aW52ZW50IGEgc2luZ2xlIG1h
Y2hpbmUgd2hpY2ggY2FuIGJl
IHVzZWQgdG8gY29tcHV0ZSBh
bnkgY29tcHV0YWJsZSBzZXF1
ZW5jZS4gSWYgdGhpcyBtYWNo
CDIS
aCBpcyB3cml0dGVuIHRoZSBT
LkQgb2Ygc29tZSBjb21wdXRp
bmcgbWFjaGluZSBNLCB0aG
VuIFUgd2lsbCBjb21wdX
RlIHRoZSBzYW1lIH
NlcXVlbmNlIG
FzIE0uCg
==