

# Software Security 1

## Administrative

---

Kevin Borgolte

[kevin.borgolte@rub.de](mailto:kevin.borgolte@rub.de)

# Tentative Lecture Schedule / Deadlines

---

## Wednesday 10-12

1. Oct 9 – Lecture
2. Oct 16 – Recitation
3. Oct 23 – Lecture
4. Oct 30 – Recitation
5. Nov 6 – Lecture
6. Nov 13 – Recitation
7. Nov 20 – Lecture
8. Nov 27 – Recitation
9. Dec 4 – Lecture
10. Dec 11 – Recitation
11. Dec 18 – Lecture
12. Jan 8 – Lecture
13. Jan 15 – Recitation
14. Jan 22 – Lecture
15. Jan 29 – Recitation

← First assignment due

← Second assignment due

← Third assignment due

← Fourth assignment due

← Fifth assignment due

← Sixth assignment due

← Seventh assignment due

# Tentative

(will probably not  
change anymore)

# Assignments

---

- Questions
  - Moodle or email us ([softsec+teaching@rub.de](mailto:softsec+teaching@rub.de))
  - If you run into issues, please report your OS, Docker version, etc.
- Assignment 6 (Holiday special)
  - 4 tasks, focusing on C++ and race conditions
  - Due: January 16th, 0:00 Bochum time
- Assignment 7
  - N tasks that will (try to) scope to be exam-sized in terms of time/difficulty
    - Please keep track of the time for them, we will be polling for it
  - Due: January 30th, 0:00 Bochum time

# Topics Today

---

- Evasys feedback
- Undefined behavior (again)
- Weird machines
- Real-world vulnerabilities
  
- Not today, but January 8th
  - Race conditions

# Software Security 1

## Evasys Feedback

---

Kevin Borgolte

[kevin.borgolte@rub.de](mailto:kevin.borgolte@rub.de)

# Feedback Recap

---

- Difficulty on the higher side, but mostly OK (-heap)
- Lack of practical examples/going too fast
  - More walkthroughs? Smaller steps? More resources?
  - Any specific topics were you found examples would help a lot more?
  - Were the recitations useful for you?
- More guidance
  - Did descriptions/task names help in getting to a solution?
  - Would it help to provide more win() functions?
  - Should we split up challenges more or did you enjoy them being a bit puzzly?

# Feedback Recap

---

- Initial ramp up difficult? Can we make it easier?
  - Linux tutorial needed? Assembler/C references needed?
  - Better/more introduction to tools/concept?
  - Or rather more resources to get started?
    - Any specific issues/parts that would help?

# Feedback Recap

---

- Thanks again for completing the Evasys survey, it helps a lot
- If you have additional feedback/concerns, please tell us
  - We can't try to do better without knowing what went wrong
    - Evasys helps, but more feedback does not hurt
- Anonymous is fine
  - e.g., retro style: Print your feedback and throw it into the mailboxes in the MC open space addressed to "Software Security MC-16"
- Non-anonymous also fine of course
- Please recap your experience of the course after lectures/after the exam



# Software Security 1

## Undefined Behavior (Again)

---

Kevin Borgolte

[kevin.borgolte@rub.de](mailto:kevin.borgolte@rub.de)

# Undefined Behavior Can Literally Erase Your Hard Disk

---

```
#include <cstdlib>

typedef int (*Function)();

static Function Do;

static int EraseAll() {
    return system("rm -rf /");
}

void NeverCalled() {
    Do = EraseAll;
}

int main() {
    return Do();
}
```

# Undefined Behavior Can Literally Erase Your Hard Disk

```
#include <cstdlib>
```

```
typedef int (*Function)();
```

```
static Function Do;
```

```
static int EraseAll() {  
    return system("rm -rf /");  
}
```

```
void NeverCalled() {  
    Do = EraseAll;  
}
```

```
int main() {  
    return Do();  
}
```

**gcc -Os -std=c++11 -Wall**

**.LC0:**

.string "rm -rf /"

EraseAll():

movl \$.LC0, %edi  
**jmp** system

NeverCalled():

movq EraseAll(), Do(%rip)  
**ret**

**main:**

**jmp** \*Do(%rip)

# Undefined Behavior Can Literally Erase Your Hard Disk

---

```
#include <cstdlib>
```

```
typedef int (*Function)();
```

```
static Function Do;
```

```
static int EraseAll() {  
    return system("rm -rf /");  
}
```

```
void NeverCalled() {  
    Do = EraseAll;  
}
```

```
int main() {  
    return Do();  
}
```

**clang -Os -std=c++11 -Wall**

```
NeverCalled():  
    ret                                # @NeverCalled()
```

```
main:  
    movl    $.L.str, %edi             # @main  
    jmp     system                    # TAILCALL
```

```
.L.str:  
    .asciz  "rm -rf /"
```

# Undefined Behavior Can Literally Erase Your Hard Disk

---

**clang -Os -std=c++11 -Wall**

```
NeverCalled():                                # @NeverCalled()
    ret

main:                                           # @main
    movl    $.L.str, %edi
    jmp     system                            # TAILCALL

.L.str:
    .asciz  "rm -rf /"
```

```
void NeverCalled() {
}

int main() {
    return system("rm -rf /");
}
```

# Undefined Behavior Can Literally Erase Your Hard Disk

---

```
#include <cstdlib>

typedef int (*Function)();

static Function Do;

static int EraseAll() {
    return system("rm -rf /");
}

void NeverCalled() {
    Do = EraseAll;
}

int main() {
    return Do();
}
```

*"If constant initialization is not performed, a variable with static storage duration or thread storage duration is zero-initialized."*

+ Using a null pointer is undefined behavior.

# Undefined Behavior Can Literally Erase Your Hard Disk

---

```
#include <cstdlib>

typedef int (*Function)();

static Function Do;

static int EraseAll() {
    return system("rm -rf /");
}

void NeverCalled() {
    Do = EraseAll;
}

int main() {
    return Do();
}
```

*"If constant initialization is not performed, a variable with static storage duration or thread storage duration is zero-initialized."*

+ Using a null pointer is undefined behavior.

1. The compiler assumes there won't be undefined behavior.

# Undefined Behavior Can Literally Erase Your Hard Disk

---

```
#include <cstdlib>

typedef int (*Function)();

static Function Do;

static int EraseAll() {
    return system("rm -rf /");
}

void NeverCalled() {
    Do = EraseAll;
}

int main() {
    return Do();
}
```

*"If constant initialization is not performed, a variable with static storage duration or thread storage duration is zero-initialized."*

+ Using a null pointer is undefined behavior.

1. The compiler assumes there won't be undefined behavior.
2. Thus, either Do() is never used or Do() is initialized.



# Undefined Behavior Can Literally Erase Your Hard Disk

---

```
#include <cstdlib>

typedef int (*Function)();

static Function Do;

static int EraseAll() {
    return system("rm -rf /");
}

void NeverCalled() {
    Do = EraseAll;
}

int main() {
    return Do();
}
```

*"If constant initialization is not performed, a variable with static storage duration or thread storage duration is zero-initialized."*

+ Using a null pointer is undefined behavior.

1. The compiler assumes there won't be undefined behavior.
2. Thus, either Do() is never used or Do() is initialized.
3. For Do() to be initialized, NeverCalled() needs to be called.

# Undefined Behavior Can Literally Erase Your Hard Disk

---

```
#include <cstdlib>

typedef int (*Function)();

static Function Do;

static int EraseAll() {
    return system("rm -rf /");
}

void NeverCalled() {
    Do = EraseAll;
}

int main() {
    return Do();
}
```

*"If constant initialization is not performed, a variable with static storage duration or thread storage duration is zero-initialized."*

+ Using a null pointer is undefined behavior.

1. The compiler assumes there won't be undefined behavior.
2. Thus, either Do() is never used or Do() is initialized.
3. For Do() to be initialized, NeverCalled() needs to be called.
4. Thus, in a well-formed program, NeverCalled() needs to be (implicitly) called before Do(), but Do() is called at the beginning.

# Undefined Behavior Can Literally Erase Your Hard Disk

```
#include <cstdlib>

typedef int (*Function)();

static Function Do;

static int EraseAll() {
    return system("rm -rf /");
}

void NeverCalled() {
    Do = EraseAll;
}

int main() {
    return Do();
}
```

*"If constant initialization is not performed, a variable with static storage duration or thread storage duration is zero-initialized."*

+ Using a null pointer is undefined behavior.

1. The compiler assumes there won't be undefined behavior.
2. Thus, either Do() is never used or Do() is initialized.
3. For Do() to be initialized, NeverCalled() needs to be called.
4. Thus, in a well-formed program, NeverCalled() needs to be (implicitly) called before Do(), but Do() is called at the beginning.
5. Considering that nothing else writes Do, it can always be set to EraseAll and we can even optimize the extra function call.

# Software Security 1

## Weird Machines

---

Kevin Borgolte

[kevin.borgolte@rub.de](mailto:kevin.borgolte@rub.de)

# Exploitation as a Core Computer Science Topic

---

- Exploitation is actually very close to core CS concepts
  - It just isn't immediately obvious how
  - Has received limited scientific/academic attention

# Exploitation as a Core Computer Science Topic

---

- Exploitation is actually very close to core CS concepts
  - It just isn't immediately obvious how
  - Has received limited scientific/academic attention
- Generally, security is computational equivalence
  - Two parsers for the same file format should accept the same language, otherwise there is a parser differential between them (that could be possible to exploit)
  - If a parser requires more than a pushdown automata (context-free language) to be implemented, then verifying equivalence is undecidable
  - Fingerprinting of systems/versions exploits non-equivalence

# Exploitation as a Core Computer Science Topic

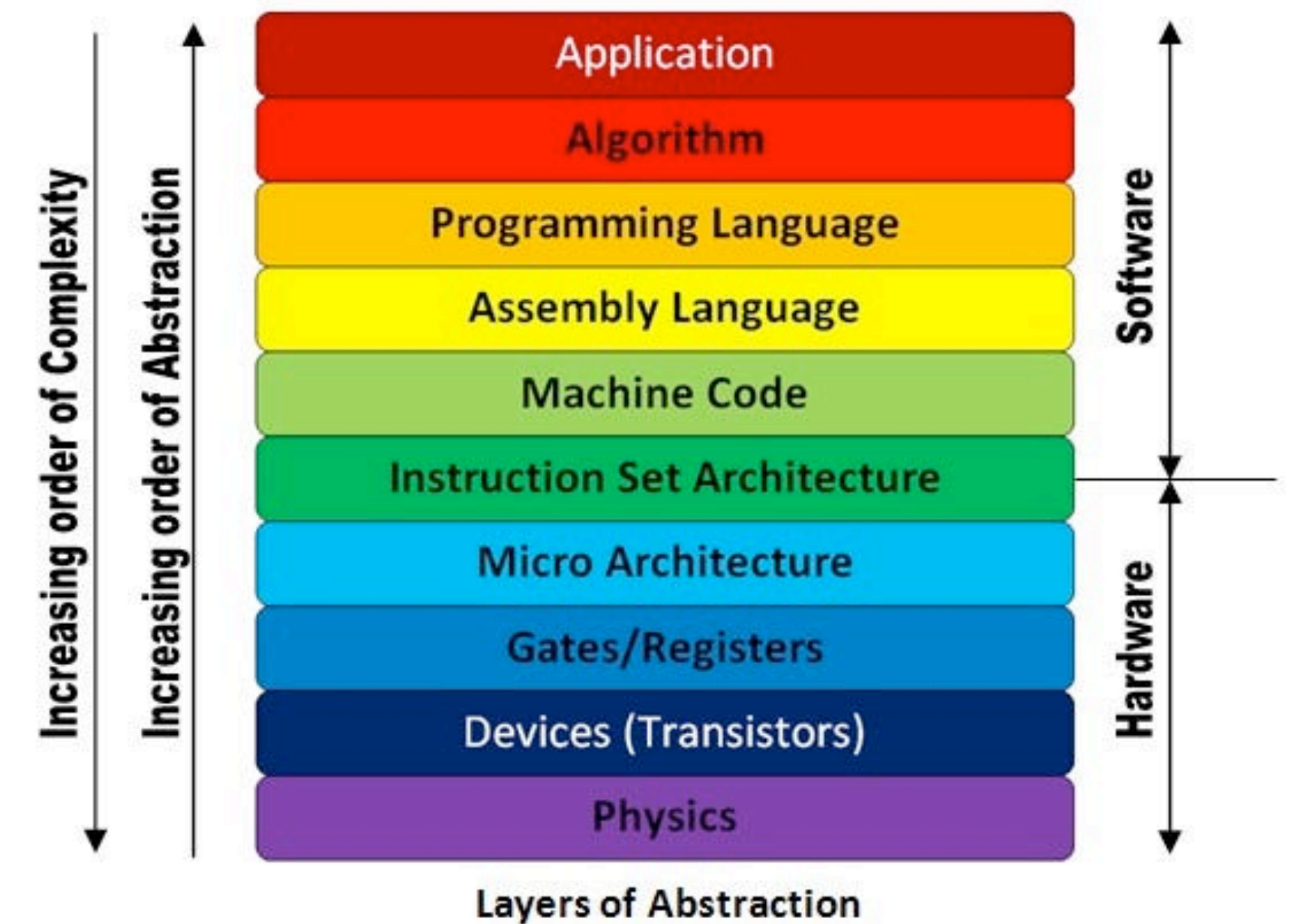
---

- Exploitation is actually very close to core CS concepts
  - It just isn't immediately obvious how
  - Has received limited scientific/academic attention
- Generally, security is computational equivalence
  - Two parsers for the same file format should accept the same language, otherwise there is a parser differential between them (that could be possible to exploit)
  - If a parser requires more than a pushdown automata (context-free language) to be implemented, then verifying equivalence is undecidable
  - Fingerprinting of systems/versions exploits non-equivalence
- Weird machines provide a theory of computation for exploitation



# Abstractions are Necessary

- Without abstractions, engineers and developers cannot specialize
  - Not needed to be a physics expert to develop applications
- One of the main driving forces of the success of computing  
<https://xkcd.com/676/>
- Problem of abstractions/division of labor: no single person knows or understands the fully stack





# Abstractions are Necessary

---

- Many parts of hardware/software engineering are about:
  - What functionality a component provides
  - What guarantees a component provides

# Abstractions are Necessary

---

- Many parts of hardware/software engineering are about:
  - What functionality a component provides
  - What guarantees a component provides
- At this border, mistakes can be introduced in many ways
  - Documenting syntactic details (=API) is easy
    - Developer can quickly use it (= get it to work sometimes)
      - On the way, they (often) make implicit assumptions and only test on some systems (but assume it works on all)
  - However, verifying it always works is an entirely different question

# Abstractions are Necessary

---

- It is no longer sufficient to rely on functional API descriptions
- Today, you need to be aware of some implementation details
  - Which details depends on circumstances
  - Examples: Kernel behavior or hardware behavior, like speculative execution or various side-channels
- This (somewhat) defeats the idea of abstractions, they leak  
See also <https://www.joelonsoftware.com/2002/11/11/the-law-of-leaky-abstractions/>
- These leaky abstractions can cause vulnerabilities/enable exploitation

# Attackers and Abstractions

---

- System components work together through their abstracted interfaces to achieve some intended goal
- Attackers do not care about this abstraction though
  - Their target is not the system as it is designed
  - Their target is the system as it is actually implemented
    - Why would we want to attack/exploit an a design? We want to subvert an actually implemented system!

# Attackers and Abstractions

---

- Like engineers, attackers need to understand the abstractions
  - However, attackers do not need to understand all abstractions
  - Attackers only need to understand the abstractions that help them subverting the system (= the weakest links)
    - Indeed, they might know system parts better than the original developers because they focused on edge cases (which the developers might have ignored because "they didn't matter")
      - For example, anticipating where a memory allocator will place memory, so that specifically crafted data can be close to each other

# Abstractions and Exploit Primitives

---

- Exploit primitives help attackers to create order in the complex implemented system
  - (Ab)using existing mechanisms and their implementation details
  - They create new building blocks that can interact with each other
  - They are abstractions on their own, just not those that the original developers intended to exist
- An exploit's small steps, reading/writing/allocating/freeing memory or syscalls, are the same as in legitimate programs
  - By chaining them differently, attackers achieve end-to-end exploitation

# Weird Machines

---

- Two views on attacker interactions
  - Developers: Inputs to the program that is processed
  - Attackers: Ways to manipulate the system's state
- What the developers consider input is code controlling the system in the view of the attacker!
  - Writing this code is considered programming a weird machine (WM)



# Weird Machines and Exploitation

---

- Exploitation as programming a WM has three stages
  - Setup: Guide the system to a flaw that can be exploited
  - Instantiation: Exploiting the flaw, to enter a state that does not conform to the specification (= weird state)
  - Execution: Transition between weird states
- Exploits can also be seen as multiple stages of weird machines, with increasing amounts of control
  - The first weird machine might only give you some control



# Weird Machines and Exploitation

---

- Before transitioning to a weird state, the attacker's view and the developer's view of program state are orthogonal
  - Developer: Higher level abstractions that implement the specification
  - Attackers: Ways to manipulate it to exploit implementation details

# Weird Machines and Exploitation

---

- Before transitioning to a weird state, the attacker's view and the developer's view of program state are orthogonal
  - Developer: Higher level abstractions that implement the specification
  - Attackers: Ways to manipulate it to exploit implementation details
- Programming weird machines can be researched and studied without there having to be a vulnerability
  - Preparation for potential bugs that may show up in the future
  - The knowledge how to program it transfers between instances of bugs

# Weird Machines and Exploitability

---

- Funnily enough: WMs introduces a new abstraction for us to reason about, which can also be modeled formally
  - Just like you have seen in Computer Science 3 (Informatik 3)
    - Finite state machines/automata
    - Turing machines and decidability
  - Intended Finite State Machine (IFSM) as the original program
  - Weird machine is a computing device over the IFSM on weird states
- This empowers us to formally prove exploitability given specific attacker capabilities, just like in cryptography
  - Cryptography: known-plaintext, chosen-plaintext attackers etc.
  - Exploitability: Arbitrary/fixed program-point + registers/no registers

# Software Security 1

## Real-world Vulnerabilities

---

Kevin Borgolte

[kevin.borgolte@rub.de](mailto:kevin.borgolte@rub.de)

# ForcedEntry

---

- Targeting Apple iOS 14 and below
  - Defeats Apple's sandbox BlastDoor
- Tracked as CVE-2021-30860
- Arbitrary remote code execution
  - Including sandbox escape
- Zero-click (requires no interaction)
  - Send an email/text message, attack happens when it is being received/parsed/shown (e.g., preview generated)

Israeli spyware firm targeted Apple devices via iMessage, researchers say

Discovery was shared with Apple, which on Monday released a patch to fix the vulnerability





# NSO Group

---

- Israeli "cyber-intelligence" company
- Known for Pegasus spyware
  - Classified as weapons/arms by the Israeli government, could not be sold without individual government approval
  - Used to target human rights activists and journalists, state espionage, warrantless surveillance, etc.
- Close to intelligence corps of Israeli defense forces (unit 8200)
- Currently being sued by basically every tech company
- On the US entity list
  - US entities cannot sell to NSO anymore



## Spyware Finally Got Scary Enough to Freak Lawmakers Out—After It Spied on Them

NSO Group's Pegasus software was used routinely to listen in on conversations with US, UK and EU officials, prompting investigations into abuses of its shockingly affordable military-grade surveillance.

By Peter Guest

January 24, 2023 at 10:00 AM GMT+1

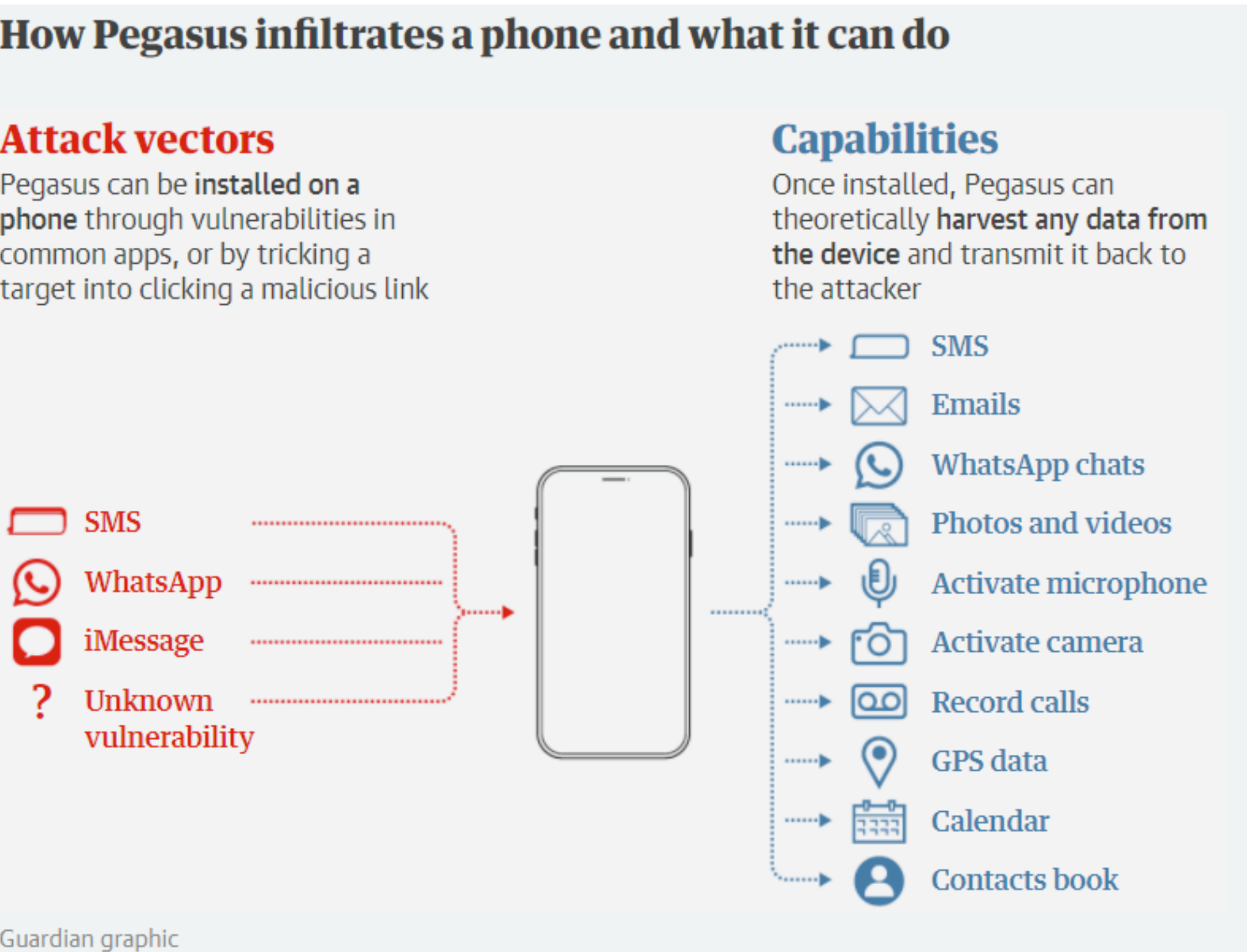
## Supreme Court clears way for WhatsApp case against NSO Group, opening spyware firm to more lawsuits

The Biden administration previously weighed in on the case between Meta and the spyware maker to recommend the court dismiss the appeal.

BY TONYA RILEY • JANUARY 9, 2023

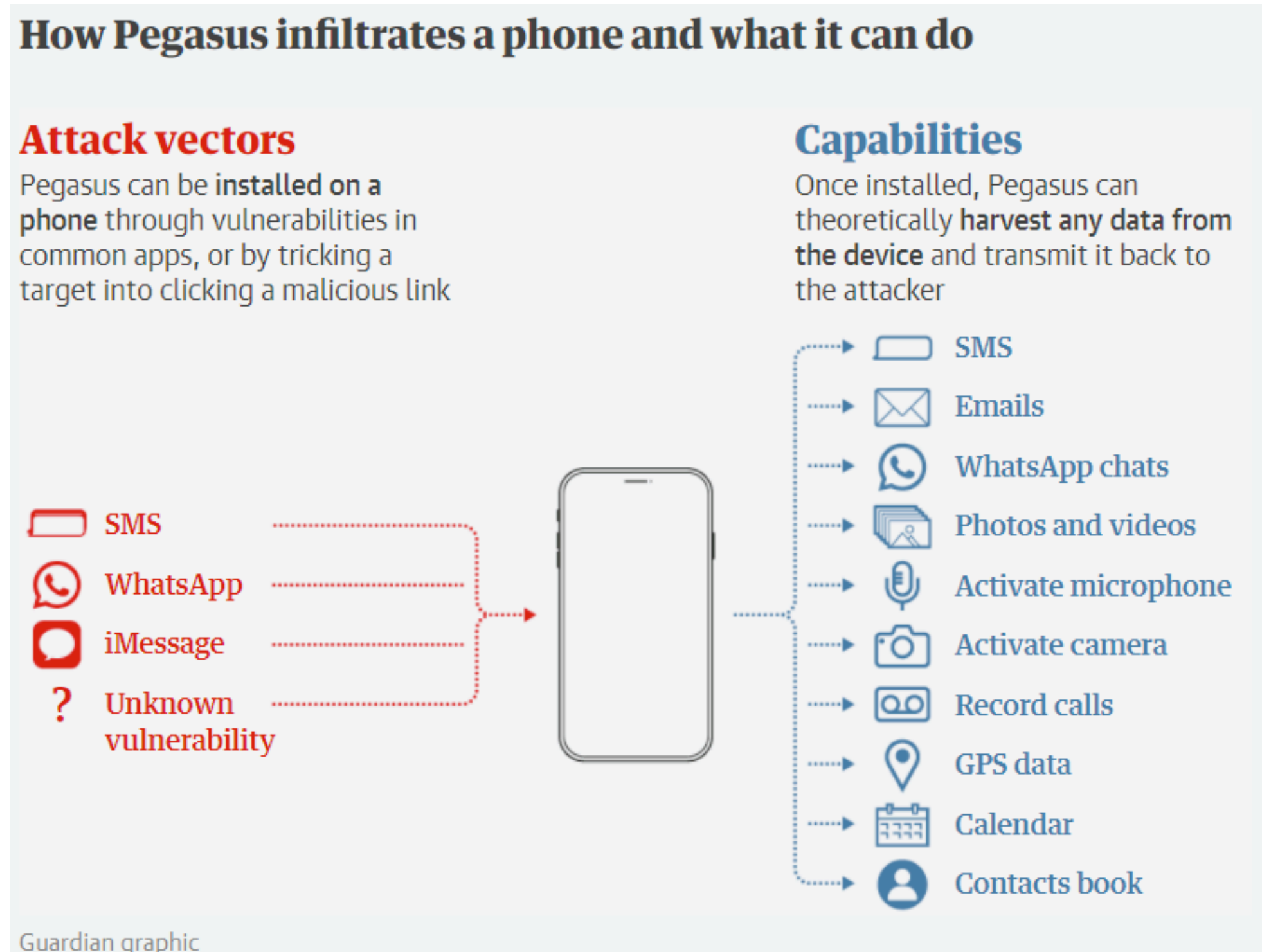
# Pegasus and ForcedEntry Level Overview

## Pegasus



# Pegasus and ForcedEntry Level Overview

## Pegasus

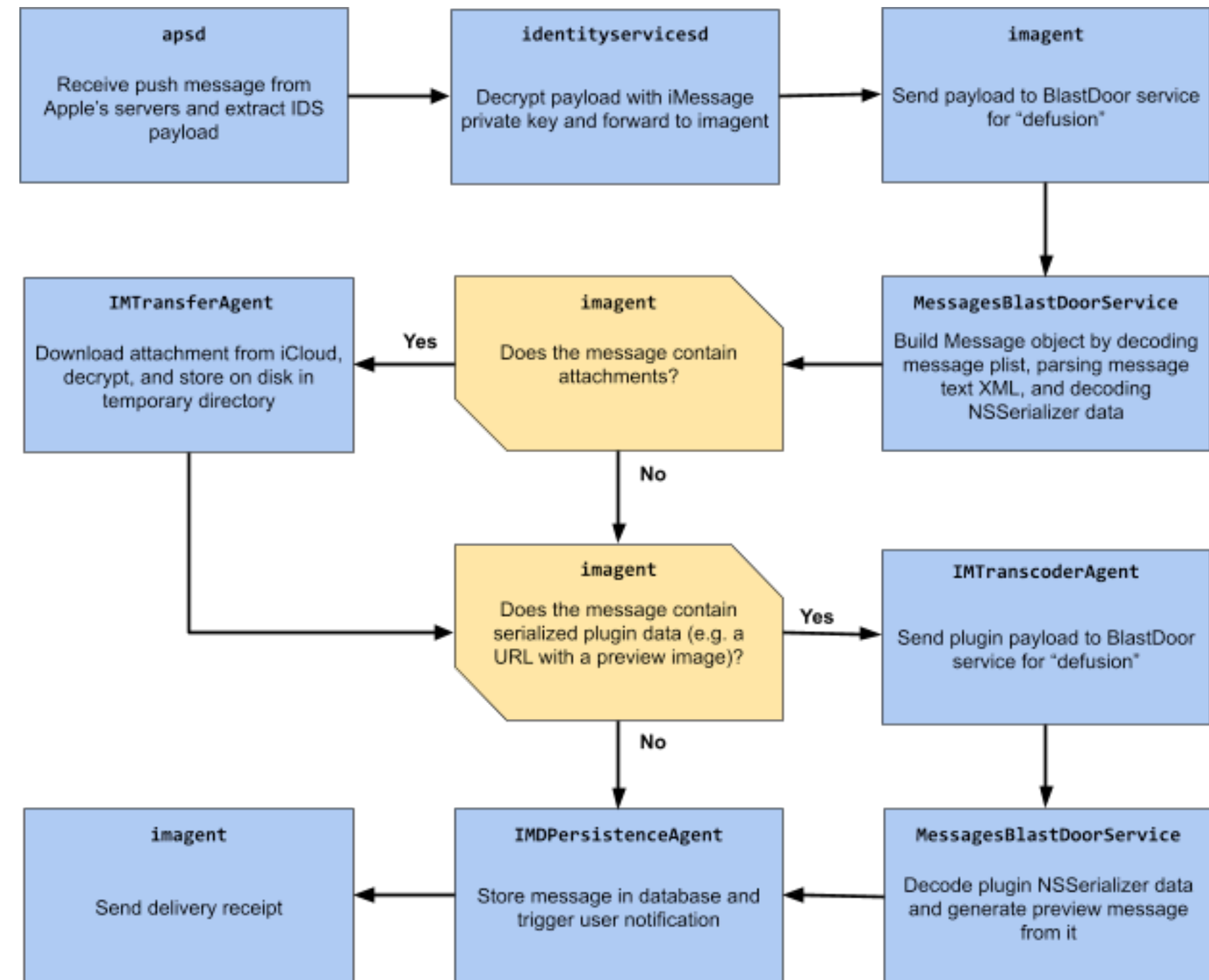


## ForcedEntry

- One exploit used in Pegasus
- Technically sophisticated
  - Known, patched, still ~\$5k-\$25k
- Approach
  - Send iMessage to victim (zero click)
  - Trigger vulnerability and gain code execution
  - Break out of sandbox

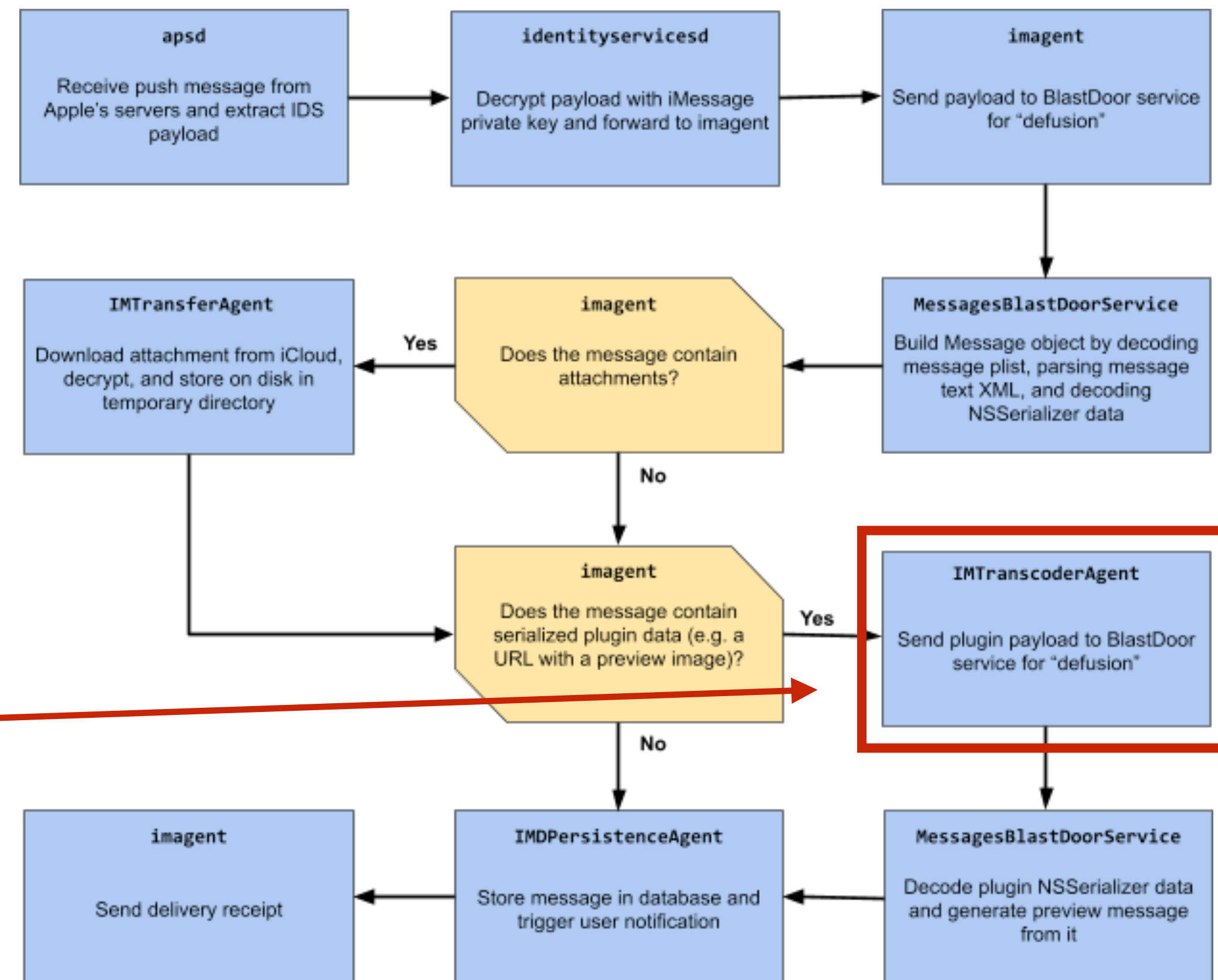


# iMessage Parsing and Processing

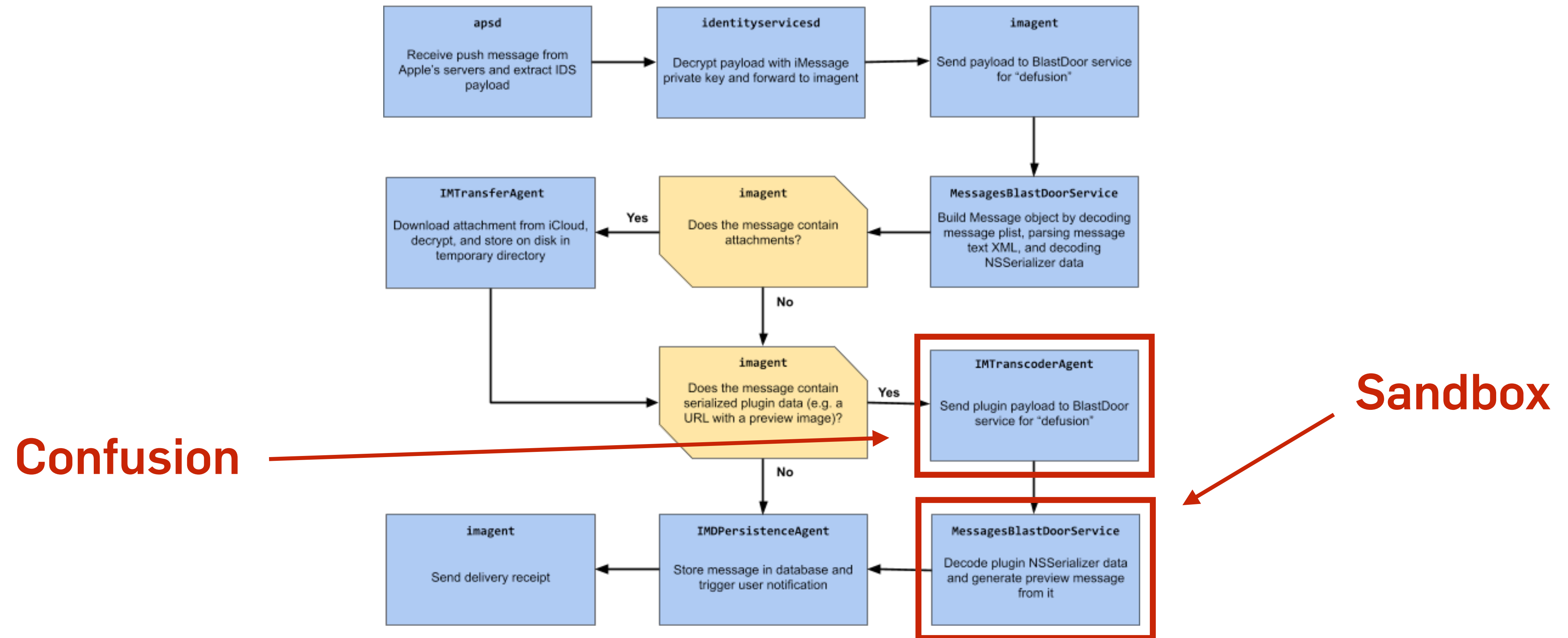


# iMessage Parsing and Processing

Confusion



# iMessage Parsing and Processing



# iMessage Parsing and Processing: Confusion

---

`[IMGIFUtils copyGifFromPath:toDestinationPath:error]`

- Special case to turn animated GIF into endless loops

# iMessage Parsing and Processing: Confusion

---

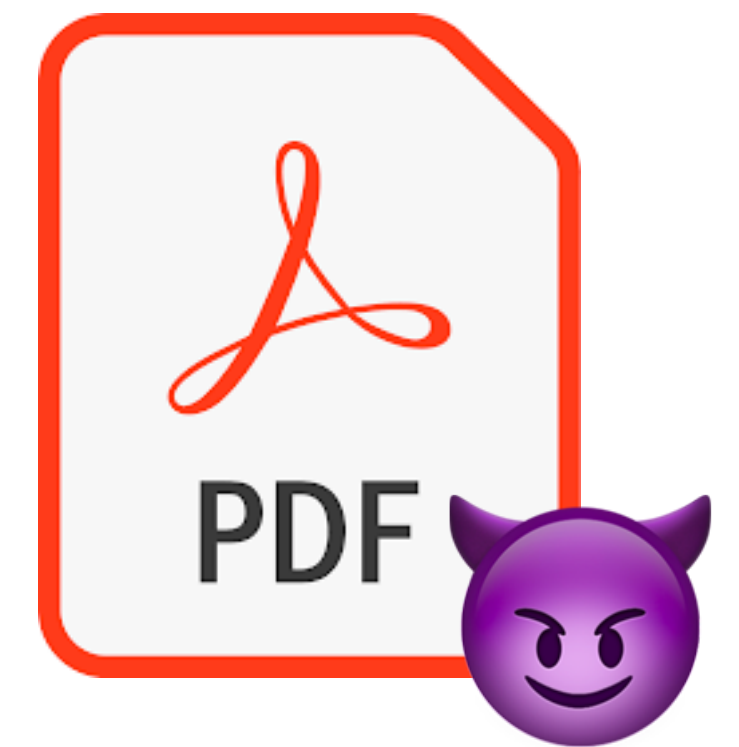
[IMGIFUtils copyGifFromPath:toDestinationPath:error]

- Special case to turn animated GIF into endless loops
- Part of Image I/O framework
  - Does not actually care whether this is a valid GIF
  - Ignores file extension completely, and guesses format
    - Usability argument: users might forget extension or use wrong one

# Image I/O Framework

---

- Ignores file extension completely, guesses format
  - We can target any file format that Image I/O supports
    - Depends on platform, up to 20+ formats
    - We can also target Adobe PDFs
- Adobe PDF
  - *Very* complicated file format, difficult to get right
  - JavaScript support → (almost) arbitrary code execution
    - CoreGraphics PDF parser does not seem to run it





# PDF Parsing?

---

- Difficult to get right, very complex
  - Old, started 1991 as Adobe-specific format
    - Now ISO with 990 pages for just PDF2.0
- Bandwidth and storage problematic early on
  - Various compression algorithms available
    - Especially targeting black/white and grayscale text compression
    - Compression algorithms by design compute and modify memory a lot

---

## ISO 32000-1:2008

Document management — Portable document format — Part 1: PDF 1.7

---

## ISO 32000-2:2020

Document management — Portable document format — Part 2: PDF 2.0

---

## ISO 32000-2:2020/DAmD 1

Document management — Portable document format — Part 2: PDF 2.0 — Amendment 1

---

## ISO/TS 32001:2022

Document management — Portable Document Format — Extensions to Hash Algorithm Support in ISO 32000-2 (PDF 2.0)

---

## ISO/TS 32002:2022

Document management — Portable Document Format — Extensions to Digital Signatures in ISO 32000-2 (PDF 2.0)

---

## ISO/DTS 32003

Document management — Portable Document Format — Adding support of AES-GCM in PDF 2.0

---

## ISO/CD TS 32004

Document management — Portable Document Format — Integrity protection in encrypted documents in PDF 2.0

---

## ISO/DTS 32005

Document management — Portable Document Format — PDF 1.7 and 2.0 structure namespace inclusion in ISO 32000-2

---

# JBIG2

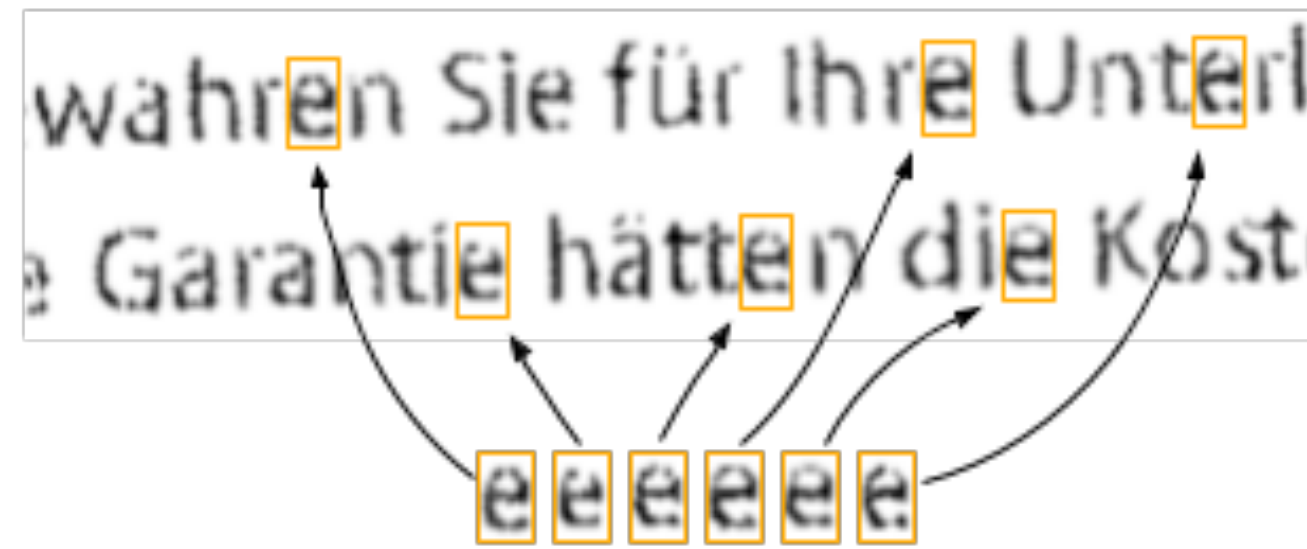
---

- JBIG2 parser vulnerability is what ForcedEntry targeted
- Bi-level (two color) image compression
- Supported by PDFs 1.4 and after
- Lossless and lossy
- Pattern matching not robust
  - Banned by BSI for the German federal government (BSI TR-03138)
  - See also [https://www.dkriesel.com/en/blog/2013/0802\\_xerox-workcentres\\_are\\_switching\\_written\\_numbers\\_when\\_scanning](https://www.dkriesel.com/en/blog/2013/0802_xerox-workcentres_are_switching_written_numbers_when_scanning)



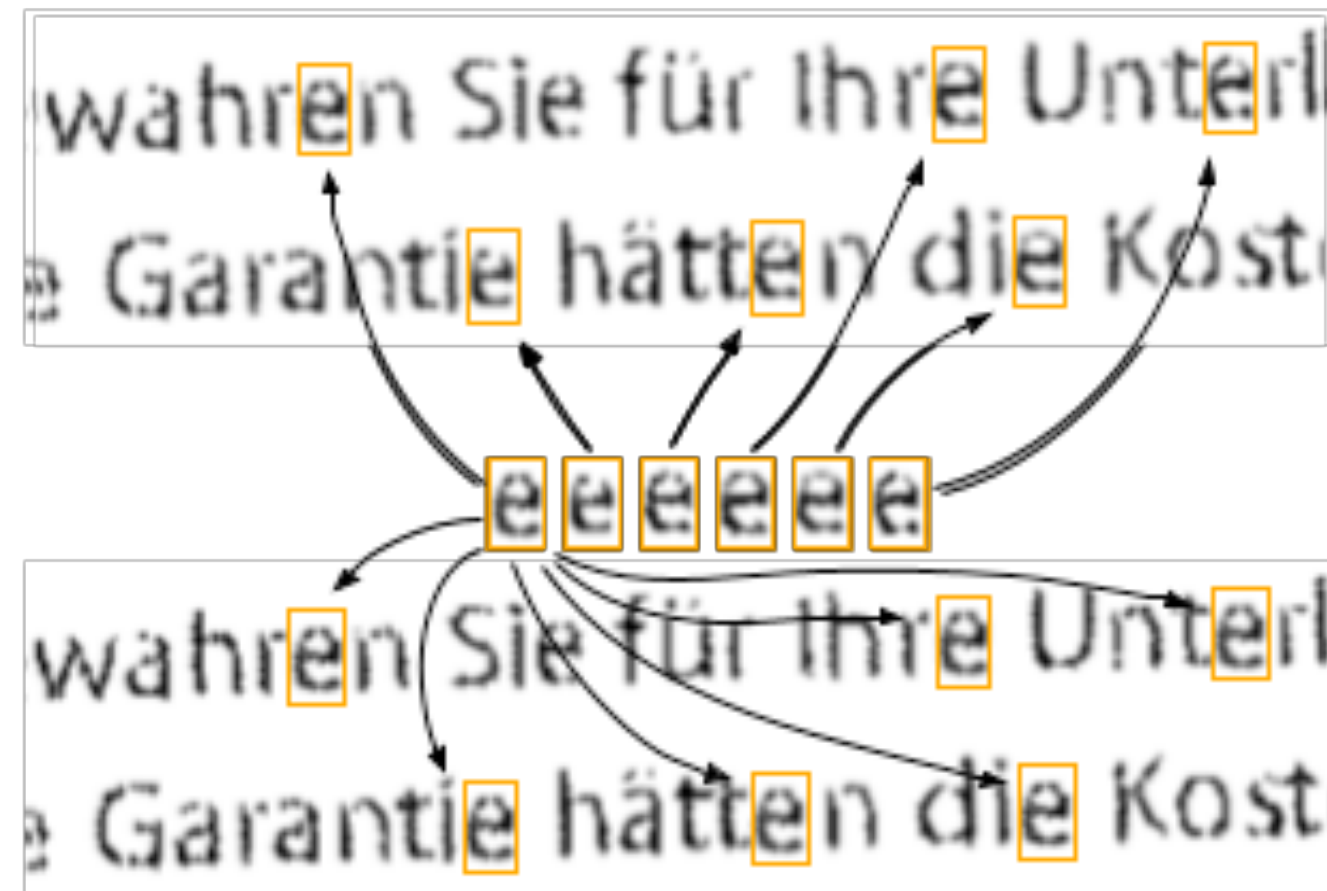
# JBIG2 Pattern Matching

---



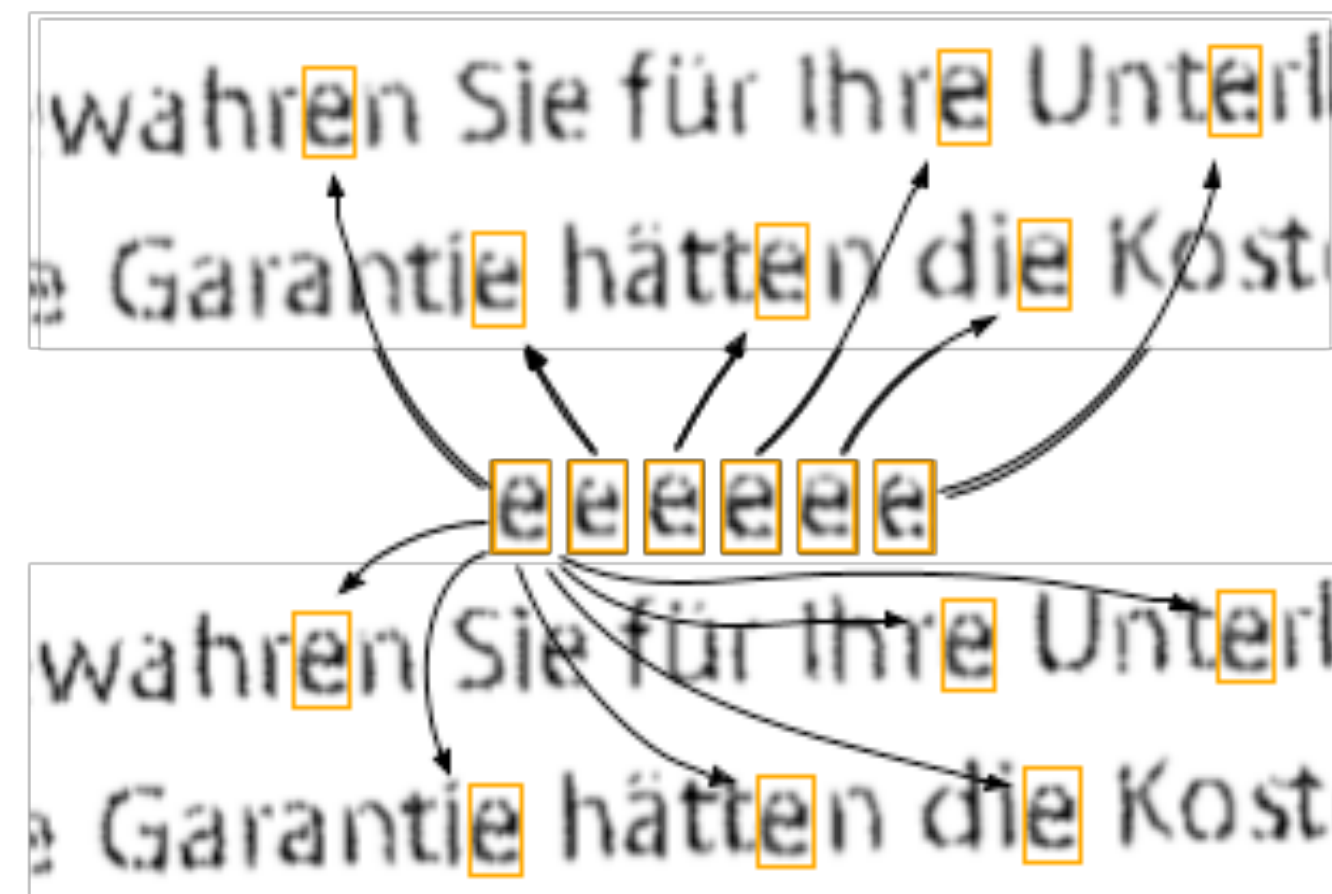
# JBIG2 Pattern Matching

---

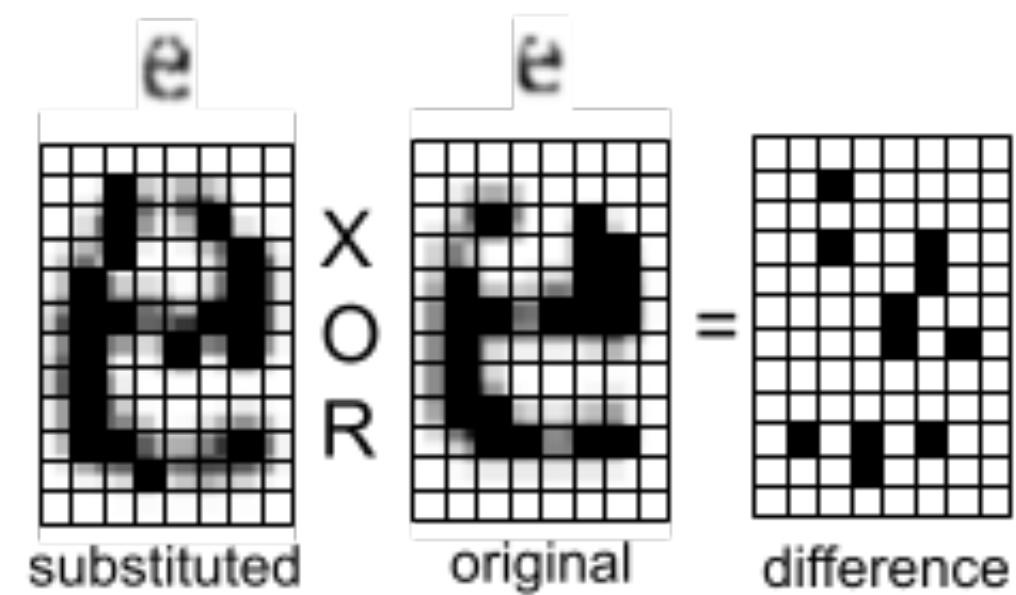
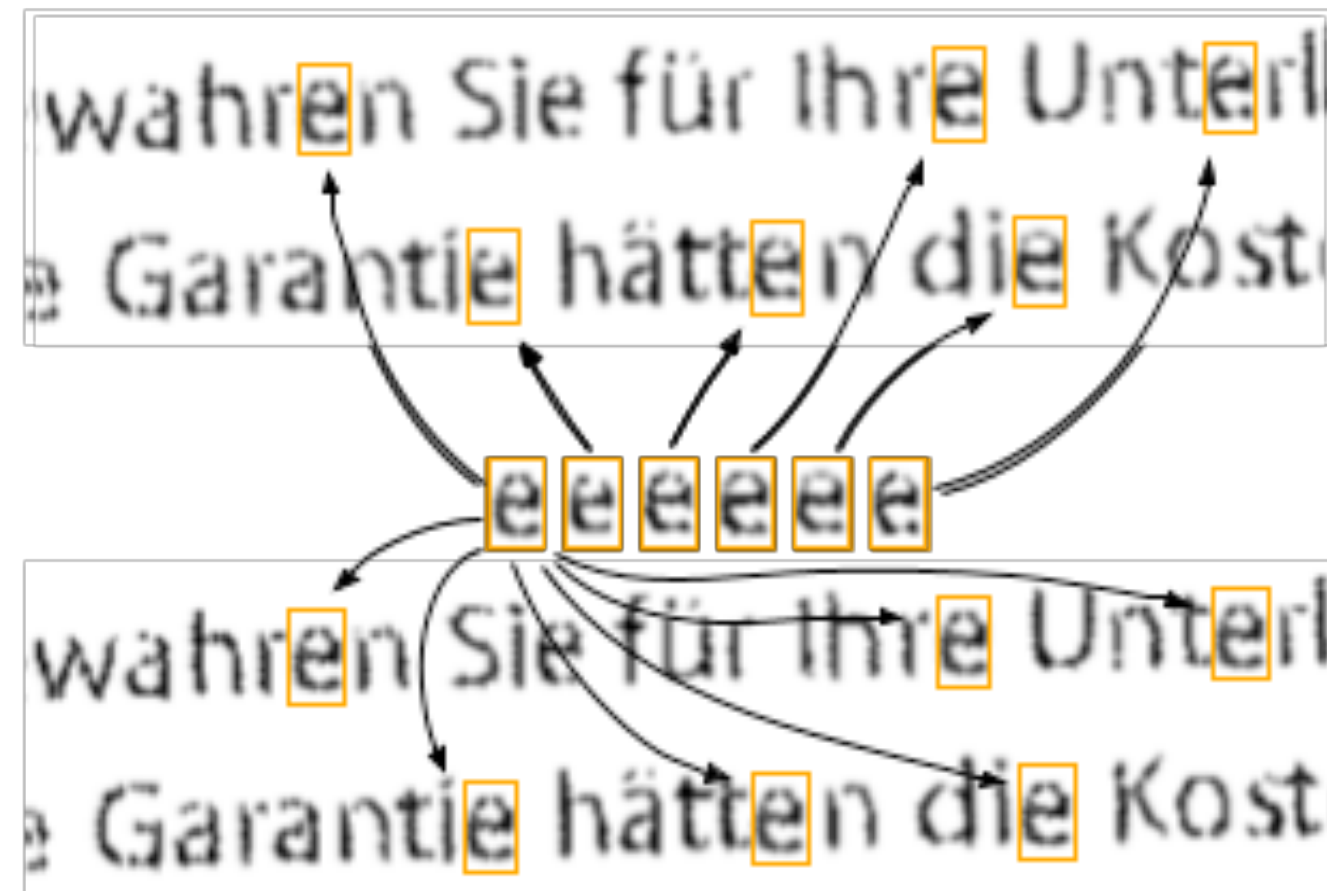


# JBIG2 Pattern Matching

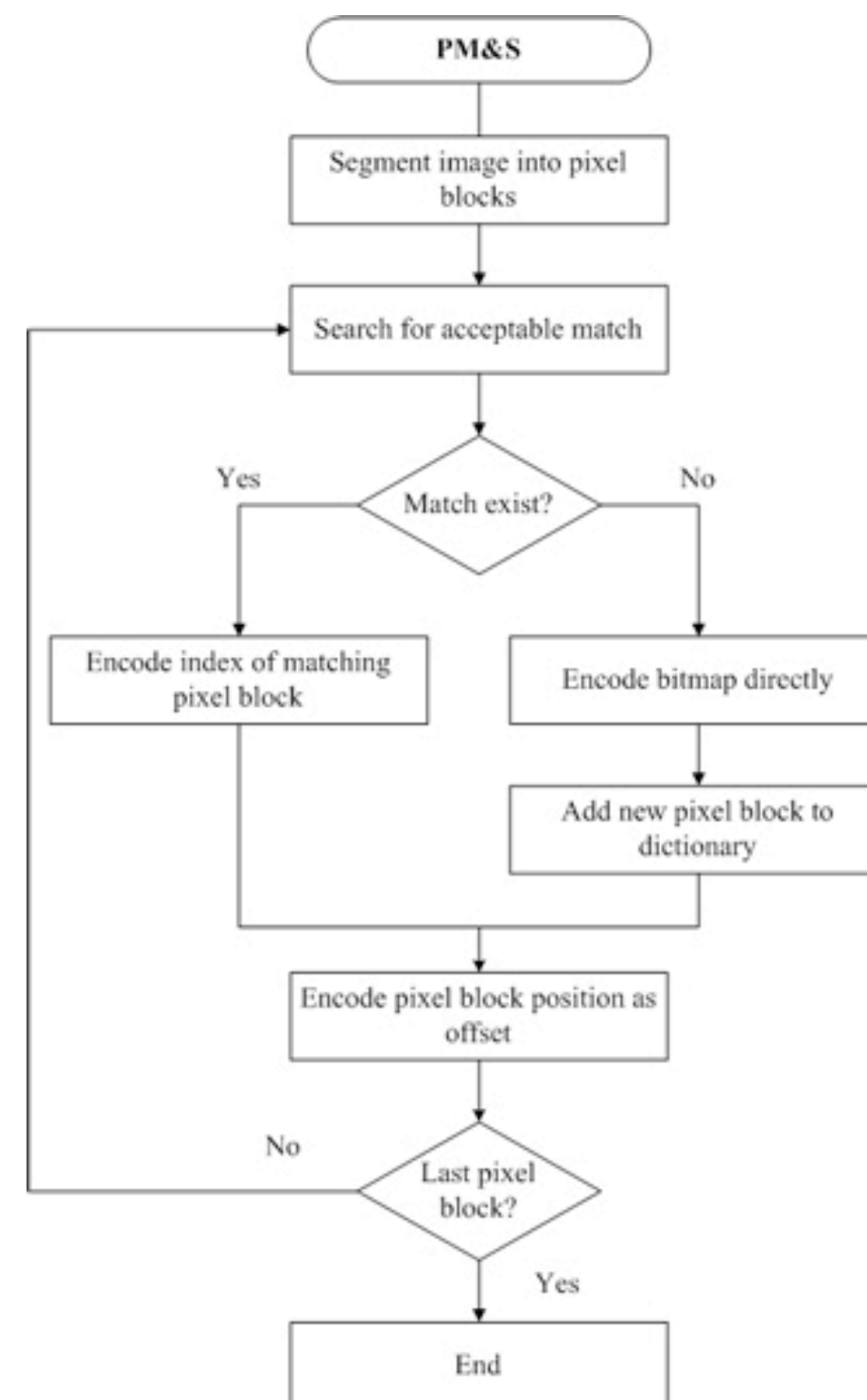
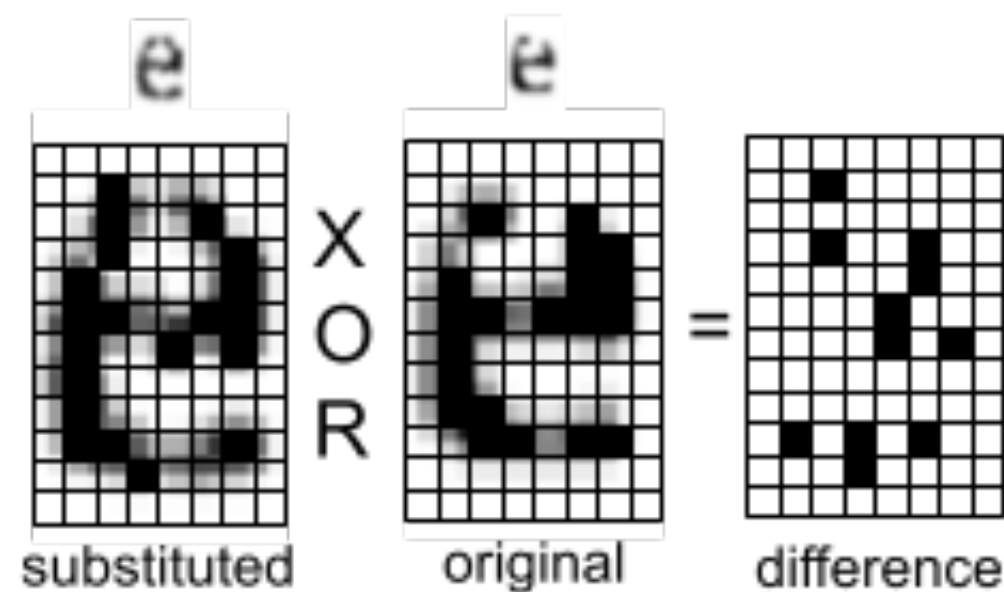
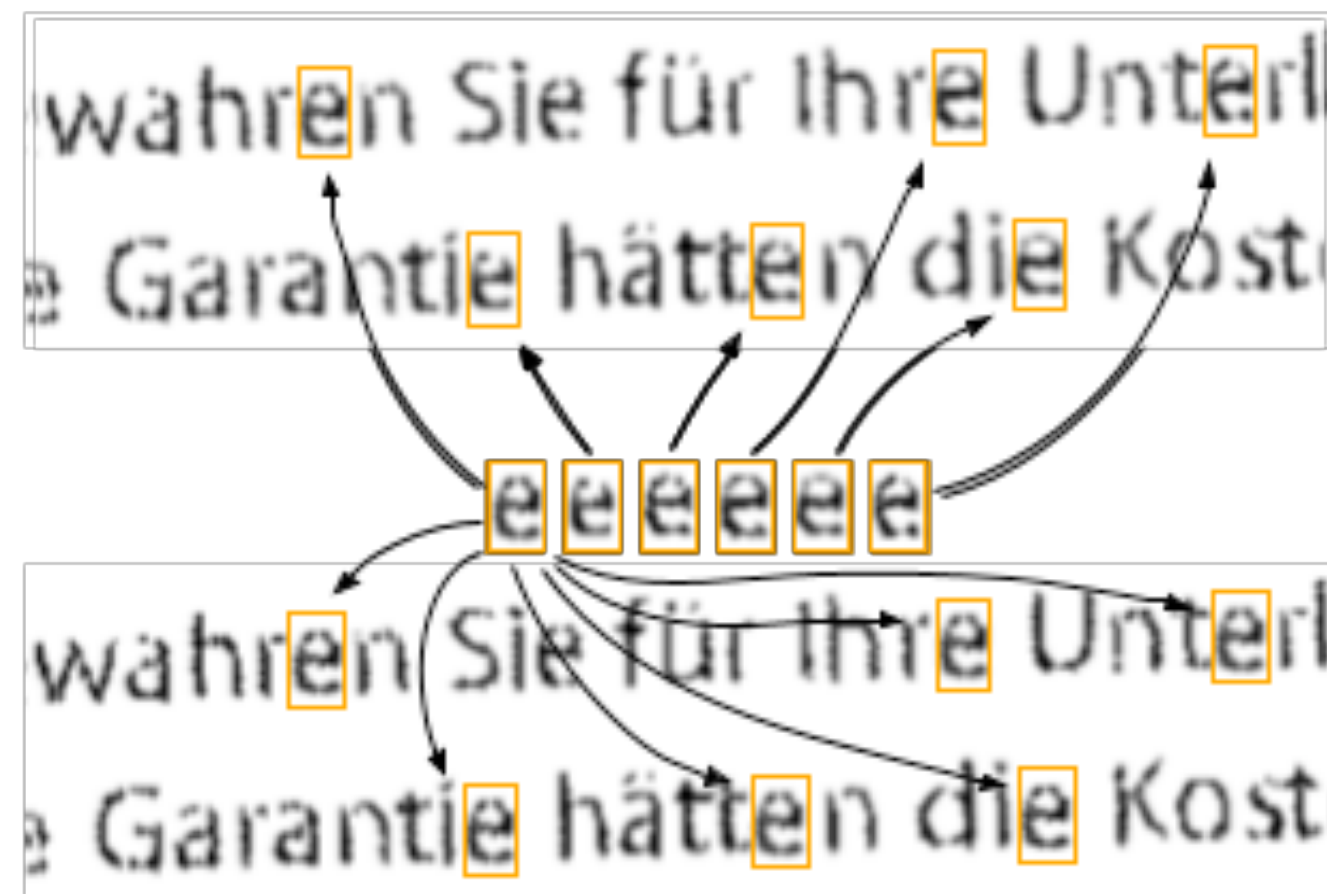
---



# JBIG2 Pattern Matching



# JBIG2 Pattern Matching



- Segment page into regions of connected pixels
  - No understanding of fonts or characters
- Encode differences of a region to its reference
  - Multiple refinements
    - Less = lossier
  - Refinements are logic ops
    - Important later on!

# ForcedEntry: JBIG2 Vulnerability

---

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

```
[...]
syms = (JBIG2Bitmap **)gmallocn(numSyms, sizeof(JBIG2Bitmap *));

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);
            }
        }
    }
}
}
```



# ForcedEntry: JBIG2 Vulnerability

---

```
Guint numSyms;
```

```
numSyms = 0;
```

```
for (i = 0; i < nRefSegs; ++i) {
```

```
    if ((seg = findSegment(refSegs[i]))) {
```

```
        if (seg->getType() == jbig2SegSymbolDict) {
```

```
            numSyms += ((JBIG2SymbolDict *)seg)->getSize();
```

```
        } else if (seg->getType() == jbig2SegCodeTable) {
```

```
            codeTables->append(seg);
```

```
        }
```

```
    } else {
```

```
        error(errSyntaxError, getPos(),
```

```
            "Invalid segment reference in JBIG2 text region");
```

```
        delete codeTables;
```

```
        return;
```

```
    }
```

```
}
```

```
[...]
```

```
[...]
```

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

```
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);
```

```
            }
```

```
        }
```

```
    }
```

```
}
```



# ForcedEntry: JBIG2 Vulnerability

**Guint numSyms;** ← **32-bit Integer**

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

```
[...]
syms = (JBIG2Bitmap **)gmallocn(numSyms, sizeof(JBIG2Bitmap *));

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);
            }
        }
    }
}
}
```

# ForcedEntry: JBIG2 Vulnerability

```
Guint numSyms;
```

```
numSyms = 0;
```

```
for (i = 0; i < nRefSegs; ++i) {
```

```
    if ((seg = findSegment(refSegs[i]))) {
```

```
        if (seg->getType() == jbig2SegSymbolDict) {
```

```
            numSyms += ((JBIG2SymbolDict *)seg)->getSize();
```

```
        } else if (seg->getType() == jbig2SegCodeTable) {
```

```
            codeTables->append(seg);
```

```
        }
```

```
    } else {
```

```
        error(errSyntaxError, getPos(),
```

```
            "Invalid segment reference in JBIG2 text region");
```

```
        delete codeTables;
```

```
        return;
```

```
    }
```

```
}
```

```
[...]
```

## Integer Overflow



```
[...]
```

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

```
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);
```

```
            }
```

```
        }
```

```
    }
```

```
}
```

# ForcedEntry: JBIG2 Vulnerability

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

**syms allocation on  
heap is too small**



```
    [...]  
    syms = (JBIG2Bitmap **)gmallocn(numSyms, sizeof(JBIG2Bitmap *));  
  
    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);  
                }  
            }  
        }  
    }  
}
```

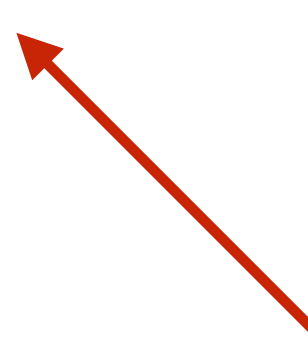
# ForcedEntry: JBIG2 Vulnerability

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

```
    [...]
```

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

    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);
                }
            }
        }
    }
}
```

 **Overwrite what follows  
syms on the heap**

# ForcedEntry: JBIG2 Vulnerability

---

```
Guint numSyms;
```

```
numSyms = 0;
```

```
for (i = 0; i < nRefSegs; ++i) {
```

```
    if ((seg = findSegment(refSegs[i]))) {
```

```
        if (seg->getType() == jbig2SegSymbolDict) {
```

```
            numSyms += ((JBIG2SymbolDict *)seg)->getSize();
```

```
        } else if (seg->getType() == jbig2SegCodeTable) {
```

```
            codeTables->append(seg);
```

```
        }
```

```
    } else {
```

```
        error(errSyntaxError, getPos(),
```

```
            "Invalid segment reference in JBIG2 text region");
```

```
        delete codeTables;
```

```
        return;
```

```
    }
```

```
}
```

```
[...]
```

```
[...]
```

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

```
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);
```

```
            }
```

```
        }
```

```
    }
```

```
}
```

# ForcedEntry: JBIG2 Integer Overflow

---

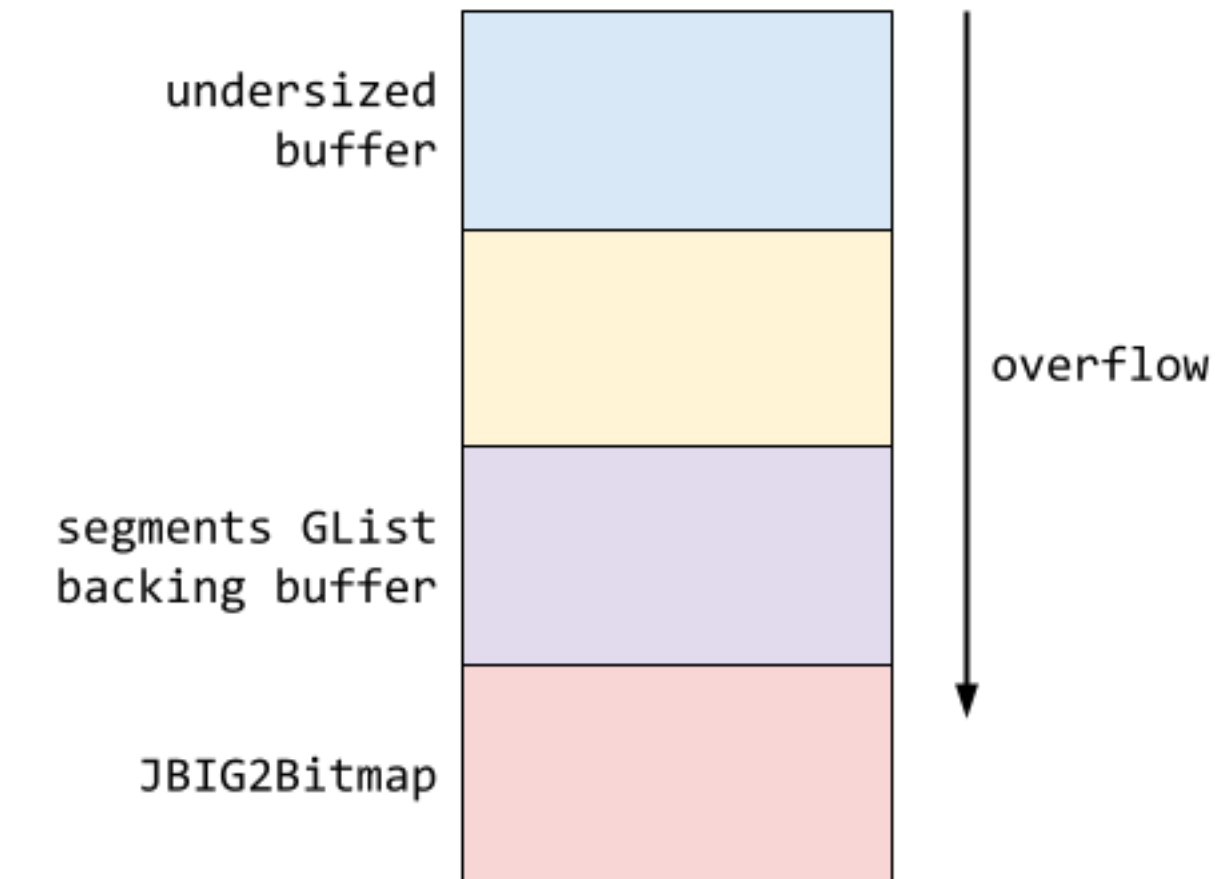
```
[...]
syms = (JBIG2Bitmap **)gmallocn(numSyms, sizeof(JBIG2Bitmap *));

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);
            }
        }
    }
}
```

- Naively, we need to overwrite a lot
  - $\text{numSyms} * \text{sizeof}(\text{JBIG2Bitmap}^*)$   
 $= \text{uint32\_max} * \text{ptr\_width}$   
 $= 32\text{GiB}$
  - Likely going to crash before anything interesting happens, and not exploitable

# ForcedEntry: JBIG2 Integer Overflow

```
[...]  
syms = (JBIG2Bitmap **)gmallocn(numSyms, sizeof(JBIG2Bitmap *));  
  
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);  
            }  
        }  
    }  
}
```

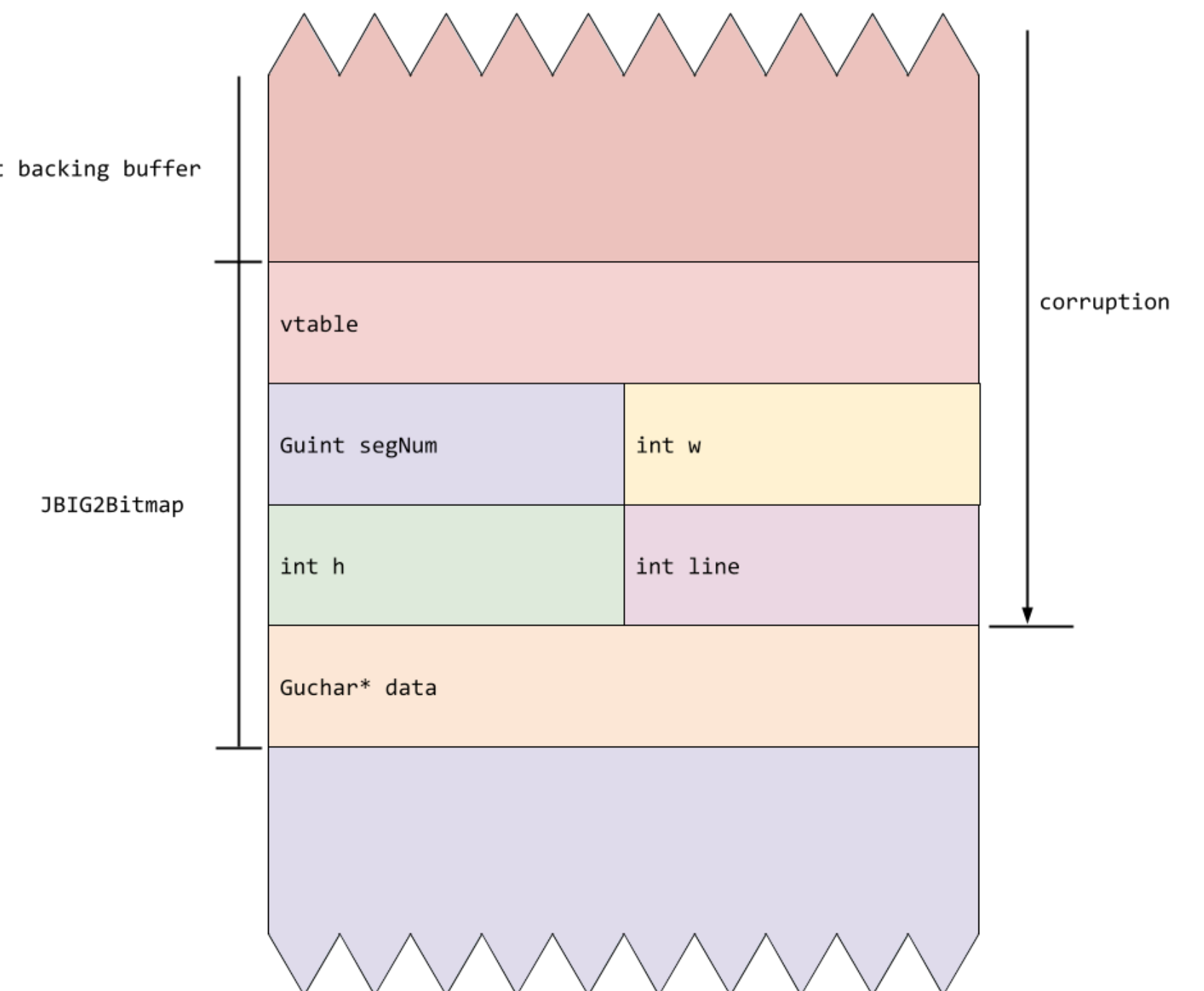


- Set up heap to
  - Control findSegment return value
    - Overwrite heap structures that are used to manage refSegs (backing buffer)
  - Misuse getType() virtual call
    - Return another type than jbig2SegSymbolDict to stop write
    - Also possible if weak pointer authentication is used because JBIG2Bitmap inherits from JBIG2Segment



# JBIG2 Integer Overflow

- The current PDF page follows the segment list in memory
  - This is what we are currently rendering and were the drawing of JBIG2 segments (characters) occurs
- We carefully craft 24 bytes (3 pointer lengths) and overwrite
  - vtable
  - segNum
  - w(idth)
  - h(eight)
  - line

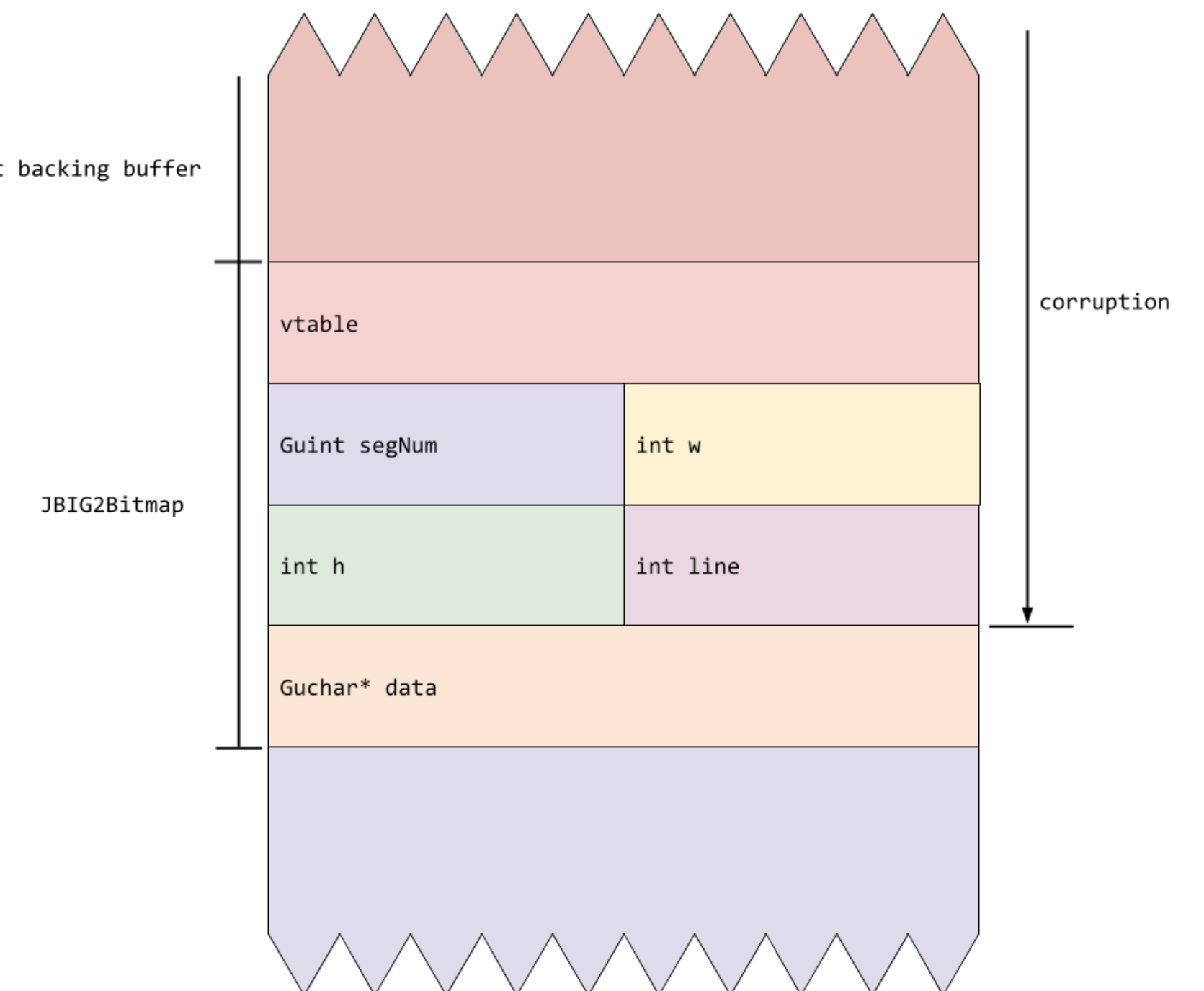


# JBIG2 Integer Overflow

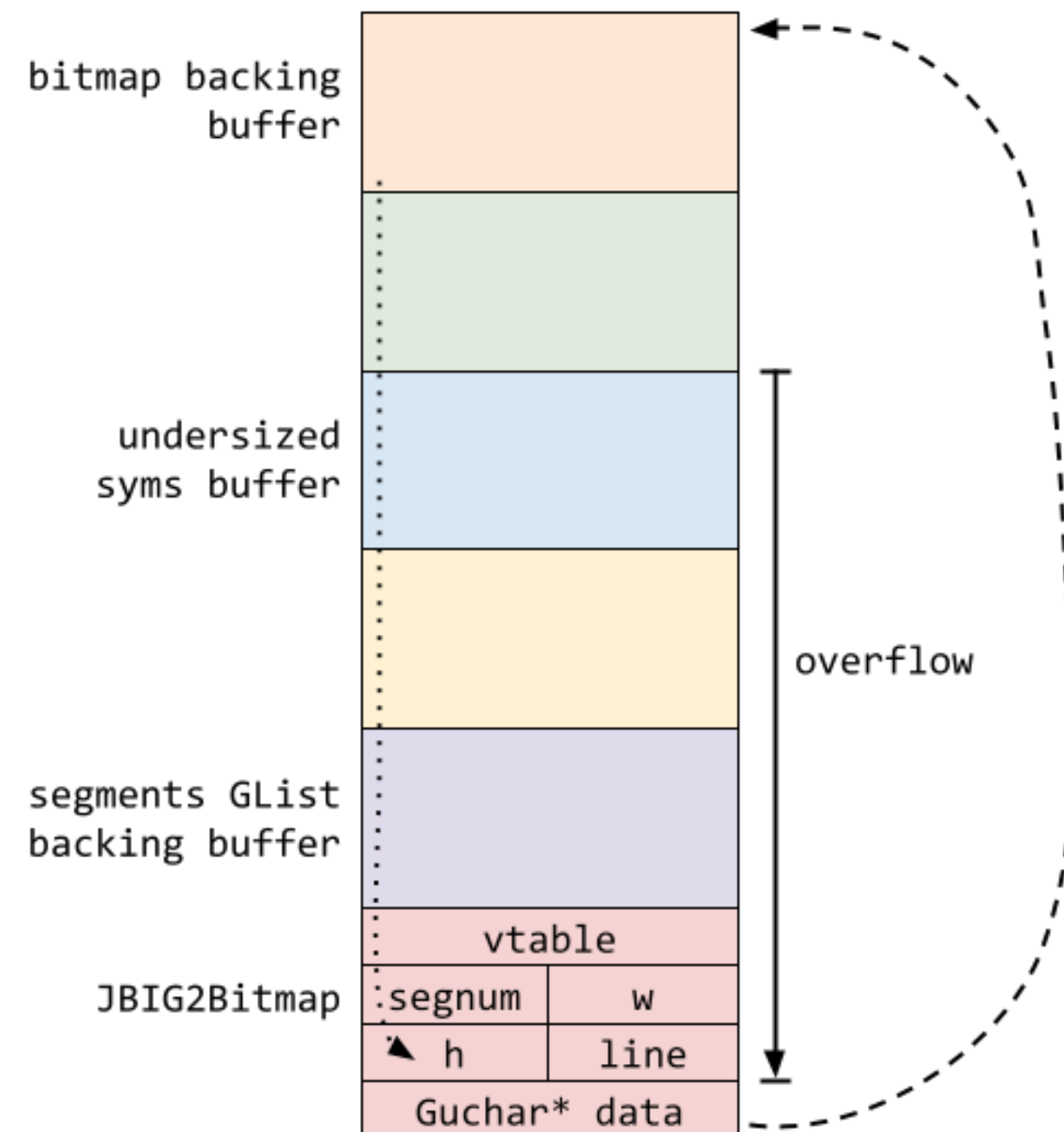
- The current PDF page follows the segment list in memory
  - This is what we are currently rendering and were the drawing of JBIG2 segments (characters) occurs
- We carefully craft 24 bytes (3 pointer lengths) and overwrite
  - vtable
  - segNum
  - w(idth)
  - h(eight)
  - line

Due to iOS architecture:

- w and line are likely 1
- segNum and h random-ish



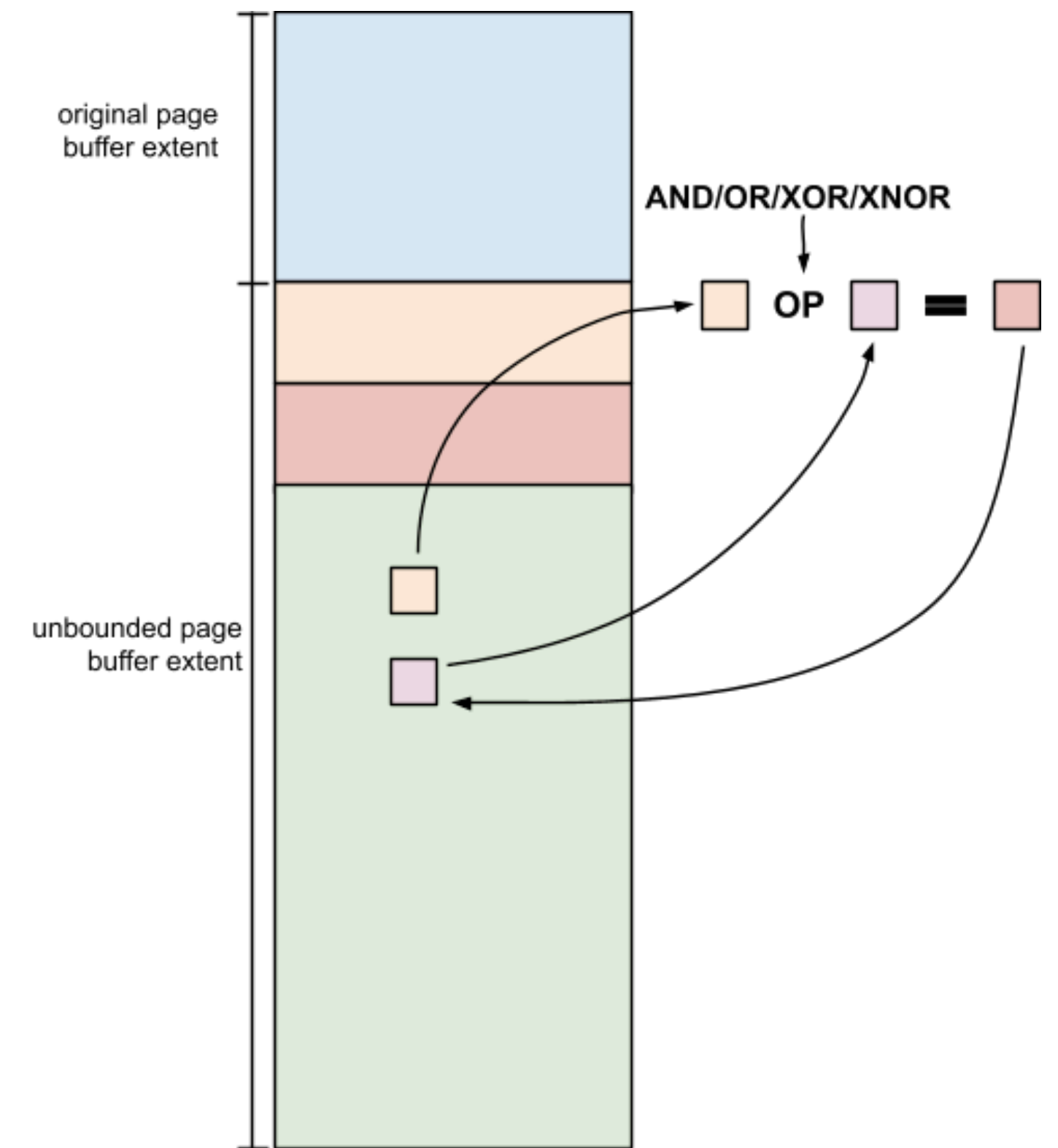
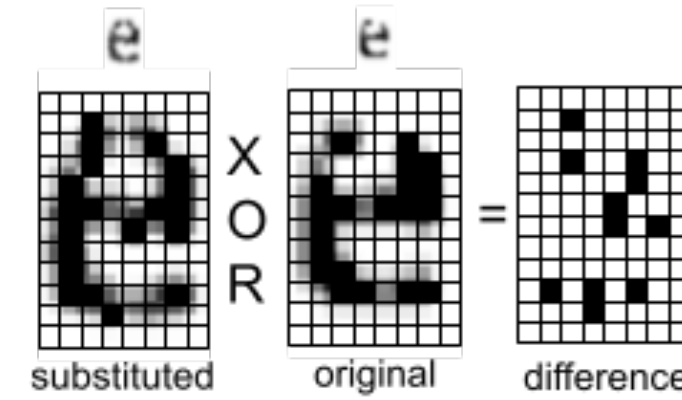
# JBIG2 Integer Overflow



- What we now have
  - Control segNum, w(idth), h(eight), line by writing a 4-byte bitmap to the PDF page
  - Arbitrary offset-based memory write by changing w, h, line of the PDF page
- What we need
  - Absolute memory address of the current PDF page on the heap, to perform arbitrary writes (defeat random heap layout, ASLR, etc.)

# JBIG2 Refinements

- Recall that refinements are just (multiple iterations) of logical operations on the segments
- We control where the parser thinks the segments are
  - Including out-of-bounds at arbitrary offsets
- We do not actually need to leak the offset, but can calculate it in memory
  - AND combined XOR is NAND, which is Turing complete



# JBIG2 Integer Overflow

---

- Overflow number of JBIG2 segments to cause a undersized allocation
- Massage heap to corrupt JBIG2 segment list stop excessive overwrite, which would otherwise lead to crash
- Overwrite current PDF page properties for offset write
- Program the weird machine to get arbitrary code execution
  - NSO Group used over 70,000 segment commands to define a small architecture within JBIG2, including their own registers, 64-bit adder, comparators etc., which they use to escape out of the sandbox

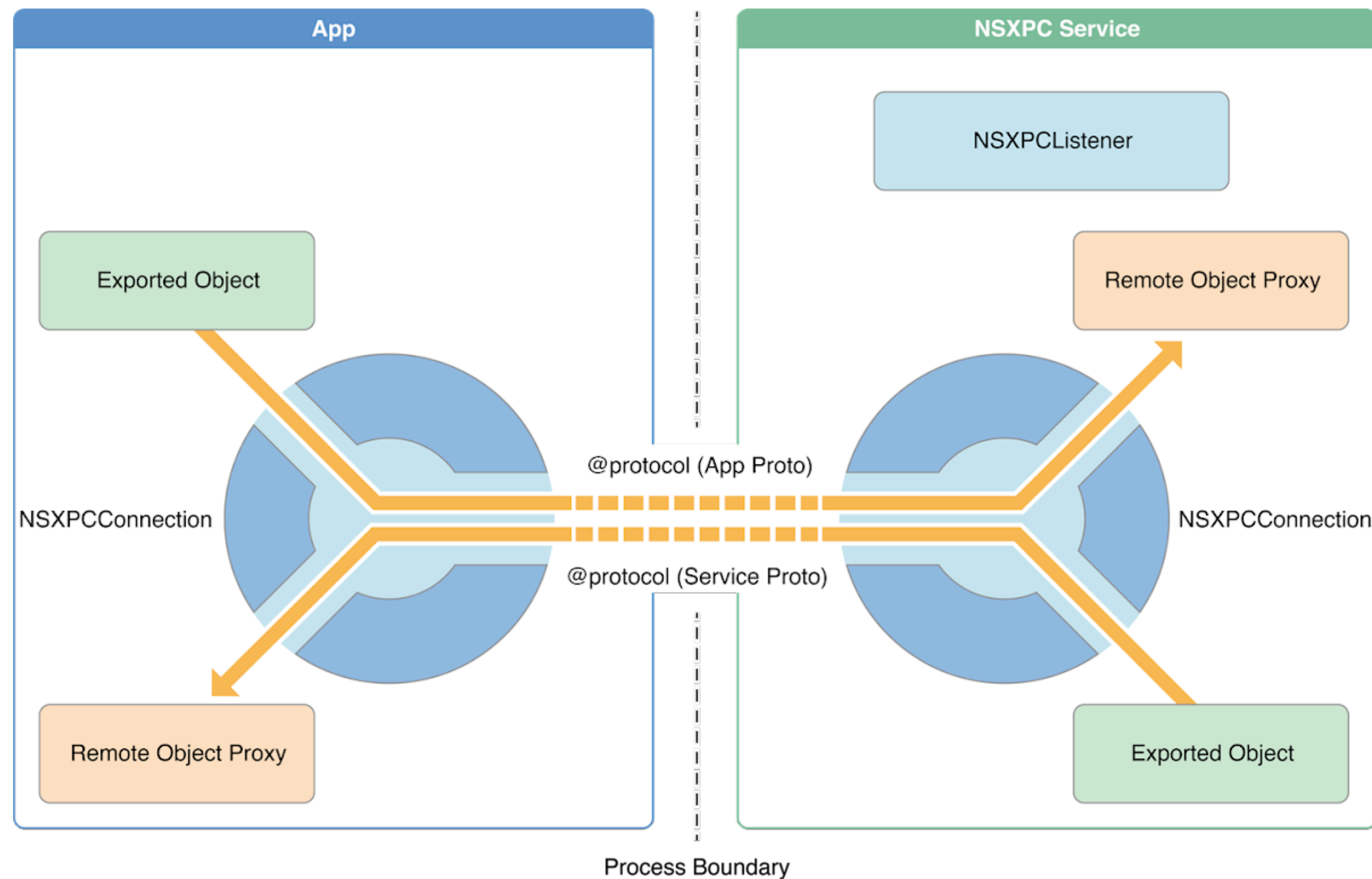
# ForcedEntry: Sandbox Escape

---

- We have arbitrary code execution, but confined to BlastDoor
- NSO Group only used logic bugs to escape
  - No native code execution, ROP, etc.
- Recall from sandboxing lecture
  - Sandboxes do not provide access to all interfaces
  - Usually, some interfaces provided by other services that enforce some security properties (e.g., remote procedure calls, RPC)



# NSXPC



- RPC mechanism for Objective C
- Semi-transparent
  - Some restrictions on what objects are allowed and objects must define what methods can be invoked and the argument types
    - Objects inheriting from an argument type are OK
- Requires serialization/deserialization



# NSXPC Objects

---

- Objects inheriting from an argument type are OK
  - Almost any Objective C class inherits from NSObject
  - If there is a RPC call that allows NSObject, we can send any object to the other process across trust boundaries, and the other process does *something* with that object

# NSXPC Objects

---

- Objects inheriting from an argument type are OK
- Almost any Objective C class inherits from NSObject
- If there is a RPC call that allows NSObject, we can send any object to the other process across trust boundaries, and the other process does *something* with that object

Frameworks/CoreTelephony.framework/CTXPCServiceSubscriberInterface-Protocol.h:39:

```
-(void)evaluateMobileSubscriberIdentity:  
    (CTXPCServiceSubscriptionContext *)arg1  
    identity:(NSObject *)arg2  
    completion:(void (^)(NSError *))arg3;
```

Frameworks/CoreTelephony.framework/CTXPCServiceCarrierBundleInterface-Protocol.h:13:

```
-(void)setWiFiCallingSettingPreferences:  
    (CTXPCServiceSubscriptionContext *)arg1  
    key:(NSString *)arg2  
    value:(NSObject *)arg3  
    completion:(void (^)(NSError *))arg4;
```

# NSPredicate NSExpression

---

- NSPredicate: "A definition of logical conditions for constraining a search for a fetch or for in-memory filtering."
- NSExpression: "An expression for use in a comparison predicate."

```
NSArray* names = @[@"one", @"two", @"three"];
```

```
NSPredicate* pred;
```

```
pred = [NSPredicate predicateWithFormat: @"SELF beginswith[c] 't'"];
```

```
NSLog(@"%@", [names filteredArrayUsingPredicate:pred]);
```

# NSPredicate NSExpression

---

- SQL-like language to filter collections of objects
- Expressions can be
  - Aggregates (in, contains, etc.)
  - Subqueries
  - Set expressions
  - Arbitrary method invocations (since 2007)  
`"FUNCTION('abc', 'stringByAppendingString', 'def')"` => `@"abcdef"`
  - Casting classes (full reflection)  
`"FUNCTION(CAST('NSFileManager', 'Class'), 'defaultManager')"`

# NSFunctionExpression

---

```
[[NSFileManager defaultManager] _web_removeFileOnlyAtPath:
 @"/tmp/com.apple.messages" stringByAppendingPathComponent:
 [ [ [ [
     [NSFileManager defaultManager]
     enumeratorAtPath: @"/tmp/com.apple.messages"
 ]
 allObjects
 ]
 filteredArrayUsingPredicate:
 [
     [NSPredicate predicateWithFormat:
         [
             [@"SELF ENDSWITH '"
                 stringByAppendingString: "XXX.gif"]
                 stringByAppendingString: "'"
             ] ] ] ]
 firstObject
 ]
 ]
 ]
```

```
NSFileManager* fm = [NSFileManager defaultManager];
NSDirectoryEnumerator* dir_enum;
dir_enum = [fm enumeratorAtPath: @"/tmp/com.apple.messages"]
NSArray* allTmpFiles = [dir_enum allObjects];
```

# NSFunctionExpression

---

```
[[NSFileManager defaultManager] _web_removeFileOnlyAtPath:
 @"/tmp/com.apple.messages" stringByAppendingPathComponent:
 [ [ [ [
     [NSFileManager defaultManager]
     enumeratorAtPath: @"/tmp/com.apple.messages"
 ]
 allObjects
 ]
 filteredArrayUsingPredicate:
 [
     [NSPredicate predicateWithFormat:
         [
             [@"SELF ENDSWITH '"
               stringByAppendingString: "XXX.gif"]
             stringByAppendingString: "'"]
         ] ] ] ]
 firstObject
 ]
 ]
 ]
```

```
NSFileManager* fm = [NSFileManager defaultManager];
NSDirectoryEnumerator* dir_enum;
dir_enum = [fm enumeratorAtPath: @"/tmp/com.apple.messages"]
NSArray* allTmpFiles = [dir_enum allObjects];
```

```
NSString* filter;
filter = [@"SELF ENDSWITH '" stringByAppendingString: "XXX.gif"];
filter = [filter stringByAppendingString: "']];
```

# NSFunctionExpression

---

```
[[[NSFileManager defaultManager] _web_removeFileOnlyAtPath:
  [@" /tmp/com.apple.messages" stringByAppendingPathComponent:
    [ [ [ [
      [NSFileManager defaultManager]
      enumeratorAtPath: @" /tmp/com.apple.messages"
    ]
    allObjects
  ]
  filteredArrayUsingPredicate:
    [
      [NSPredicate predicateWithFormat:
        [
          [@"SELF ENDSWITH '"
            stringByAppendingString: "XXX.gif"]
            stringByAppendingString: "'"]
        ] ] ] ]
  firstObject
]
]
]
```

```
NSFileManager* fm = [NSFileManager defaultManager];
NSDirectoryEnumerator* dir_enum;
dir_enum = [fm enumeratorAtPath: @" /tmp/com.apple.messages"]
NSArray* allTmpFiles = [dir_enum allObjects];

NSString* filter;
filter = [@"SELF ENDSWITH '" stringByAppendingString: "XXX.gif"];
filter = [filter stringByAppendingString: "'"];
```

```
NSPredicate* pred;
pred = [NSPredicate predicateWithFormat: filter]
```



# NSFunctionExpression

---

```
[[NSFileManager defaultManager] _web_removeFileOnlyAtPath:
 @"/tmp/com.apple.messages" stringByAppendingPathComponent:
 [ [ [ [
     [NSFileManager defaultManager]
     enumeratorAtPath: @"/tmp/com.apple.messages"
 ]
 allObjects
 ]
 filteredArrayUsingPredicate:
 [
     [NSPredicate predicateWithFormat:
     [
         [@"SELF ENDSWITH '"
         stringByAppendingString: "XXX.gif"]
         stringByAppendingString: "'"]
     ] ] ] ]
 firstObject
 ]
 ]
 ]
```

```
NSFileManager* fm = [NSFileManager defaultManager];
NSDirectoryEnumerator* dir_enum;
dir_enum = [fm enumeratorAtPath: @"/tmp/com.apple.messages"]
NSArray* allTmpFiles = [dir_enum allObjects];

NSString* filter;
filter = [@"SELF ENDSWITH '" stringByAppendingString: "XXX.gif"];
filter = [filter stringByAppendingString: "'"];

NSPredicate* pred;
pred = [NSPredicate predicateWithFormat: filter]

NSArray* matches;
matches = [allTmpFiles filteredArrayUsingPredicate: pred];
```

# NSFunctionExpression

---

```
[[NSFileManager defaultManager] _web_removeFileOnlyAtPath:
 @"/tmp/com.apple.messages" stringByAppendingPathComponent:
 [ [ [ [
     [NSFileManager defaultManager]
     enumeratorAtPath: @"/tmp/com.apple.messages"
 ]
 allObjects
 ]
 filteredArrayUsingPredicate:
 [
     [NSPredicate predicateWithFormat:
         [
             [@"SELF ENDSWITH '"
                 stringByAppendingString: "XXX.gif"]
             stringByAppendingString: "'"
         ] ] ] ]
firstObject
 ]
 ]
 ]
```

```
NSFileManager* fm = [NSFileManager defaultManager];
NSDirectoryEnumerator* dir_enum;
dir_enum = [fm enumeratorAtPath: @"/tmp/com.apple.messages"]
NSArray* allTmpFiles = [dir_enum allObjects];

NSString* filter;
filter = [@"SELF ENDSWITH '" stringByAppendingString: "XXX.gif"];
filter = [filter stringByAppendingString: "'"];

NSPredicate* pred;
pred = [NSPredicate predicateWithFormat: filter]

NSArray* matches;
matches = [allTmpFiles filteredArrayUsingPredicate: pred];

NSString* gif_subpath = [matches firstObject];
```

# NSFunctionExpression

```
[[NSFileManager defaultManager] _web_removeFileOnlyAtPath:
    @"/tmp/com.apple.messages" stringByAppendingPathComponent:
    [ [ [ [
        [NSFileManager defaultManager]
        enumeratorAtPath: @"/tmp/com.apple.messages"
    ]
    allObjects
    ]
    filteredArrayUsingPredicate:
    [
        [NSPredicate predicateWithFormat:
            [
                [@"SELF ENDSWITH '"
                stringByAppendingString: "XXX.gif"]
                stringByAppendingString: "'"]
            ] ] ] ]
    firstObject
    ]
    ]
    ]
```

```
NSFileManager* fm = [NSFileManager defaultManager];
NSDirectoryEnumerator* dir_enum;
dir_enum = [fm enumeratorAtPath: @"/tmp/com.apple.messages"]
NSArray* allTmpFiles = [dir_enum allObjects];

NSString* filter;
filter = [@"SELF ENDSWITH '" stringByAppendingString: "XXX.gif"];
filter = [filter stringByAppendingString: "'"];

NSPredicate* pred;
pred = [NSPredicate predicateWithFormat: filter]

NSArray* matches;
matches = [allTmpFiles filteredArrayUsingPredicate: pred];

NSString* gif_subpath = [matches firstObject];

NSString* root = @"/tmp/com.apple.messages";
NSString* full_path;
full_path = [root stringByAppendingPathComponent: gifSubpath];
```

# NSFunctionExpression

```
[[NSFileManager defaultManager] _web_removeFileOnlyAtPath:
 @"/tmp/com.apple.messages" stringByAppendingPathComponent:
 [ [ [ [
     [NSFileManager defaultManager]
     enumeratorAtPath: @"/tmp/com.apple.messages"
 ]
 allObjects
 ]
 filteredArrayUsingPredicate:
 [
     [NSPredicate predicateWithFormat:
         [
             [@"SELF ENDSWITH '"
                 stringByAppendingString: "XXX.gif"]
             stringByAppendingString: "'"]
         ] ] ] ]
 firstObject
 ]
 ]
```

```
NSFileManager* fm = [NSFileManager defaultManager];
NSDirectoryEnumerator* dir_enum;
dir_enum = [fm enumeratorAtPath: @"/tmp/com.apple.messages"]
NSArray* allTmpFiles = [dir_enum allObjects];

NSString* filter;
filter = [@"SELF ENDSWITH '" stringByAppendingString: "XXX.gif"];
filter = [filter stringByAppendingString: "'"];

NSPredicate* pred;
pred = [NSPredicate predicateWithFormat: filter]

NSArray* matches;
matches = [allTmpFiles filteredArrayUsingPredicate: pred];

NSString* gif_subpath = [matches firstObject];

NSString* root = @"/tmp/com.apple.messages";
NSString* full_path;
full_path = [root stringByAppendingPathComponent: gifSubpath];

NSFileManager* fm = [NSFileManager defaultManager];
[fm _web_removeFileOnlyAtPath: full_path];
```

This is how NSO deleted the file from disk.

# Sandbox Escape: Idea

---

- Create a NSPredicate object
- Pass it as NSObject to a privileged process via NSXPC
  - Here: CommCenter providing CoreTelephonyClient
  - Get it deserialized and evaluate the predicate

# Sandbox Escape: Idea

---

- Create a NSPredicate object
- Pass it as NSObject to a privileged process via NSXPC
  - Here: CommCenter providing CoreTelephonyClient
  - Get it deserialized and evaluate the predicate
- Small issue: Evaluation does not happen on deserialization
  - NSPredicate is only only parsed, not evaluated
  - Evaluation requires explicit call to allowEvaluation() method
  - Not a typical workflow for RPC and security sensitive



# ForcedEntry: Sandbox Escape

---

- Find an object that invokes allowEvaluation on its predicate
  - PTRow in Apple's private PrototypeTools framework does so
- Problem: PrototypeTools is not loaded in CommCenter
  - Solution: Send a serialized object via NSXPC that depends itself on PrototypeTools and forces CommCenter to load PrototypeTools
- Similar to SeLector Oriented Programming (SLOP)  
<https://bugs.chromium.org/p/project-zero/issues/detail?id=1933>

# ForcedEntry

---

- JBIG2 weird machine to gain arbitrary code execution
  - File format confusion and integer overflow
- Leverage logic vulnerabilities in exposed RPC functionality to break out of BlastDoor sandbox
  - Serialization of untrusted input in a trusted environment
- Mitigations
  - Certain operations with significant side effects prohibited in NSExpression (e.g., creating and destroying objects) since iOS15
  - Generally, input validation and only allowing narrow sets of what is allowed

# Software Security 1

## Outlook

---

Kevin Borgolte

[kevin.borgolte@rub.de](mailto:kevin.borgolte@rub.de)

# Summer Semester 2025

---

- Lecture: Software Security 2 (5 CP) (requires softsec1)
  - Similar to softsec1, but more advanced topics, like JIT, sandboxing, browsers, kernel and hypervisors, non-x86 architectures, non-Linux OS, etc.
  - Wednesday 10–12 Lectures/Recitations and Thursday 12–14 Exercise
- Lecture: Program Analysis (5 CP) (no prerequisites)
  - Methodological approach to program understanding, analysis guarantees (e.g., can this buffer overflow?), program/exploit synthesis etc.
  - Thursdays 10–12 Lectures/Recitations and Tuesday 10–12 Exercise
- Research Projects (4 CP each)
  - Initial Research in Software Security (Bachelor)
  - Research in Software Security (Master)
  - also "in Internet Security"
- Seminar: Software and Internet Security (3 CP)

# Open Positions/Theses

---

- We have quite a few open positions/theses right now
  - Student assistants (SHK/WHB)
  - Master theses (mainly; limited Bachelor theses, but do ask)
  - PhD researcher positions
- Primary topics (non-exclusive list!)
  - Program analysis/understanding/fuzzing and automatic vulnerability discovery/exploitation for
    - Data-only attacks/mitigations
    - Stateful programs/applications
  - (Automated) network protocol analysis (from implementations)
  - Large-scale network measurements

# Paid Student Assistantships

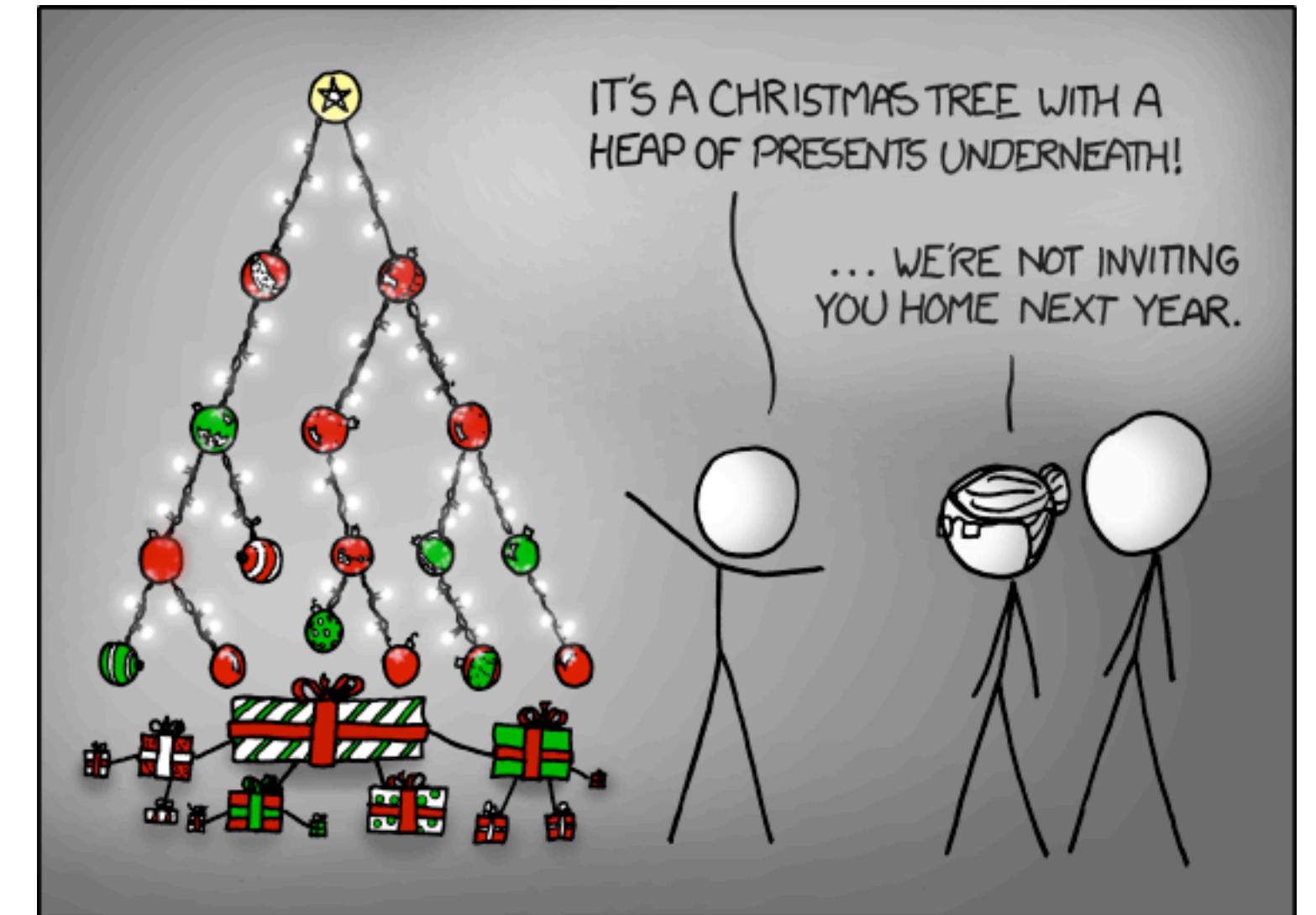
---

- Right now, we are especially looking for
  - Teaching support (starting ASAP)
    - Software Security 1: Writing new challenges
    - Software Security 2: Writing new challenges
    - Program Analysis: Implementing a data-flow/abstract interpretation scaffold (preferably in Rust or Modern C++)
  - Teaching assistants
    - Software Security 2 (starting next semester)
    - Program Analysis (starting next semester)
    - Software Security 1 (starting next winter)
  - Research assistants (primarily on fuzzing, program analysis, and network measurements)



# Paid Student Assistantships

- Right now, we are especially looking for
  - Teaching support (starting ASAP)
    - Software Security 1: Writing new challenges
    - Software Security 2: Writing new challenges
    - Program Analysis: Implementing a data-flow/abstract interpretation scaffold (preferably in Rust or Modern C++)
  - Teaching assistants
    - Software Security 2 (starting next semester)
    - Program Analysis (starting next semester)
    - Software Security 1 (starting next winter)
  - Research assistants (primarily on fuzzing, program analysis, and network measurements)



## Happy Holidays!