Let me explain how this undermines integrity verification with a specific example:

Suppose we're using this naive concatenation approach to verify the integrity of files in a system. An attacker could:

1. Original legitimate files:

```
/etc/passwd
/etc/shadow
```

When concatenated: "/etc/passwd/etc/shadow"

1. Malicious manipulation:

```
/etc/pas
/swd/etc/shadow
```

When concatenated: "/etc/pas" + "swd/etc/shadow" = "/etc/passwd/etc/shadow"

Both would produce the SAME hash value, even though:

- In case 1, we have two valid system files
- In case 2, we have two different files, potentially hiding malicious content

This undermines integrity because:

1. We cannot detect this tampering
2. The hash is supposed to change if the data changes in any way
3. The system would incorrectly verify these as "identical" when they're actually different

This is why proper integrity verification needs to account for boundaries between objects, usually by either:

- Hashing objects separately
- Including length prefixes
- Using unambiguous delimiters