

# [CRITICAL] SSL/TLS Certificate Verification Bypass via Insecure Defaults

## Summary

The implementation of the `OpenSSL::SSL::SSLContext` class contains a critical security flaw where the constructor (`initialize`) explicitly disables all forms of certificate and hostname verification. This overrides secure `DEFAULT_PARAMS` and forces the application to trust any certificate presented during a TLS handshake, including self-signed or malicious certificates.

## Technical Details

```
...\openssl\lib\openssl\ssl.rb
def initialize(version = nil)
  self.ssl_version = version if version
  self.verify_mode = OpenSSL::SSL::VERIFY_NONE # Vulnerability: Disables peer
verification
  self.verify_hostname = false # Vulnerability: Disables hostname
matching
end
```

By setting `VERIFY_NONE`, the application skips the validation of the certificate chain. By setting `verify_hostname` to `false`, it fails to ensure that the certificate belongs to the server it is connecting to.

### Impact

This vulnerability facilitates a **Man-in-the-Middle (MitM)** attack. An attacker positioned on the same network or with control over DNS can:

1. **Interception:** Decrypt and read sensitive traffic (API keys, credentials, PII) in plaintext.
2. **Manipulation:** Modify requests or server responses without the application's knowledge.
3. **Data Theft:** Fully compromise account sessions and backend integrations.

## Proof of Concept (PoC)

### 1. Attacker Setup (Listener)

A malicious server was created using a self-signed certificate to mimic `api.secure-bank.com`.

```
# Attacker.rb snippet
ssl_context.cert = self_signed_cert
ssl_context.key = private_key
ssl_server = OpenSSL::SSL::SSLServer.new(TCPServer.new(4444), ssl_context)
```

## 2. Victim Execution

The victim script, using the vulnerable SSLContext logic, connected to the attacker.

```
# Victim.rb snippet
ctx.verify_mode = OpenSSL::SSL::VERIFY_NONE
# Connection succeeds despite invalid certificate
```

## 3. Result (Captured Data)

The attacker successfully captured the following "encrypted" payload in plaintext:

**Captured Header:** POST /login HTTP/1.1

**Captured JSON Body:** {"user": "victim", "pass": "BountyHunter2026"}

## ARP Spoofing exploit/DNS Spoofing exploit/

The exploit relies on Layer 2 (Data Link) vulnerability to achieve a Layer 7 (Application) data breach. The application's job is to protect data even if the network is compromised.

While modern network hardware can implement 'Dynamic ARP Inspection' (DAI) to prevent the redirection of traffic, the application's failure to verify SSL certificates ensures that even in 'secure' environments, a single compromised node or a malicious DNS entry can lead to total credential exposure. If the router itself is hacked, DAI won't help because the router is the one doing the redirecting. If

a user is tricked into using a malicious proxy server (WPAD attack), the Ruby script will connect through it and bypass security..

## Recommended Remediation

ruby openssl library in current github repository. ...\\openssl\\lib\\openssl\\ssl.rb

```
def initialize(version = nil)
  self.set_params(DEFAULT_PARAMS) # Use defined secure defaults
  self.verify_mode = OpenSSL::SSL::VERIFY_PEER
  self.verify_hostname = true
  self.cert_store = OpenSSL::X509::Store.new
  self.cert_store.set_default_paths
  self.ssl_version = version if version
end
```

## Severity

- **Severity:** Critical (8.1)
- **CVSS:** CVSS:3.1/AV:N/AC:H/PR:N/UI:N/S:U/C:H/I:H/A:H