nullhat 2025: SSS

@elesquina

The service implements a Schnorr-like signature over DSA parameters (p, q, g) with public key $y = g^x \bmod p$. A signature is (s, e) where $r = g^k \bmod p$, $e = \text{SHA256}(r \parallel m)$ truncated to q-bytes, and $s \equiv k - xe \pmod q$. Verification recomputes $r_v \equiv g^s y^e \pmod p$ and checks that the truncated hash of $r_v \parallel m$ equals the provided e.

When registering a planet, the server generates a DSA key, computes a fingerprint as the first two bytes of SHA256(public_key_PEM), and stores a signature for the message ``` ``expedition started'' ``` concatenated with that fingerprint. For the flag, the client sends a planet name, a public key, and a signature for ``` ``this planet is good'' ``` concatenated with the fingerprint of the submitted public key. The critical bug is that the server verifies the stored planet signature using the public key that the client provides, instead of the planet's real public key, and the fingerprint is only two bytes.

Let the stored signature be (s_1, e_1) and let the original public key returned at registration be (p, q, g_0, y_0). Under that key, the verifier computes $r \equiv g_0^{s_1} y_0^{e_1} \bmod p$, and by construction $e_1 = H(r \parallel m_1)$ for $m_1 =$ ``` ``expedition started'' ``` $\parallel$ `fp` where `fp` is the planet fingerprint. We can build a different public key that makes the same (s_1, e_1) produce the same r during verification. Choose an integer a with $\gcd(a, q) = 1$, set $g_1 \equiv r^a \pmod p$, and define

$$x_1 \equiv (a^{-1} - s_1)\, e_1^{-1} \pmod q.$$

With $y_1 = g_1^{x_1} \bmod p$, verification using (p, q, g_1, y_1) yields $g_1^{s_1} y_1^{e_1} \equiv r \pmod p$, so it reproduces the exact r that was hashed during registration, and the stored signature validates under the attacker-chosen key.

The remaining obstacle is the fingerprint, but it is only two bytes, so a brute force collision is feasible. We iterate over a, construct the corresponding key (p, q, g_1, y_1), export it to PEM, and stop when SHA256(PEM)[: 2] matches the planet fingerprint. At that point, submitting this swapped-in public key passes the server's first verification step. We also know x_1, so we can sign $m_2 =$ ``` ``this planet is good'' ``` $\parallel$ `fp` and submit that signature to retrieve the flag from the service.