

Where Did I Leave My Keys?

Lessons from the Juniper Dual EC Incident

By Stephen Checkoway, Jacob Maskiewicz, Christina Garman, Joshua Fried, Shaanan Cohney, Matthew Green, Nadia Heninger, Ralf-Philipp Weinmann, Eric Rescorla, and Hovav Shacham

Abstract

In December 2015, Juniper Networks announced multiple security vulnerabilities stemming from unauthorized code in ScreenOS, the operating system for their NetScreen Virtual Private Network (VPN) routers. The more sophisticated of these vulnerabilities was a passive VPN decryption capability, enabled by a change to one of the parameters used by the Dual Elliptic Curve (EC) pseudorandom number generator.

In this paper, we described the results of a full independent analysis of the ScreenOS randomness and VPN key establishment protocol subsystems, which we carried out in response to this incident. While Dual EC is known to be insecure against an attacker who can choose the elliptic curve parameters, Juniper had claimed in 2013 that ScreenOS included countermeasures against this type of attack. We find that, contrary to Juniper's public statements, the ScreenOS VPN implementation has been vulnerable to passive exploitation by an attacker who selects the Dual EC curve point since 2008. This vulnerability arises due to flaws in Juniper's countermeasures as well as a cluster of changes that were all introduced concurrently with the inclusion of Dual EC in a single 2008 release. We demonstrate the vulnerability on a real NetScreen device by modifying the firmware to install our own parameters, and we show that it is possible to passively decrypt an individual VPN session in isolation without observing any other network traffic. This incident is an important example of how guidelines for random number generation, engineering, and validation can fail in practice. Additionally, it casts further doubt on the practicality of designing a safe "exceptional access" or "key escrow" scheme of the type contemplated by law enforcement agencies in the United States and elsewhere.

1. INTRODUCTION

In December 2015, Juniper announced that an "internal code review" revealed the presence of "unauthorized code in ScreenOS that could allow a knowledgeable attacker [...] to decrypt VPN connections." In response to this, Juniper released patched versions of ScreenOS, the operating system powering the affected NetScreen devices, but has declined to disclose any further information about the intrusion and vulnerability.

Immediately following Juniper's advisory, security researchers around the world—including our team—began examining the ScreenOS firmware to find the vulnerabilities Juniper had patched. They found that the change that rendered ScreenOS encryption breakable did

nothing but replace a few embedded constants in Juniper's pseudorandom number generator. The reason why this results in an attacker being able to decrypt connections is Juniper's design decision to use the NSA-designed Dual EC Pseudorandom Number Generator (PRNG).^{4,12} Dual EC has the problematic property that an attacker who knows the discrete logarithm of one of the input parameters (Q) with respect to a generator point, and is able to observe a small number of consecutive bytes from the PRNG, can then compute the internal state of the generator and thus predict all future output. Thus, it is critical that the discrete logarithm of Q remain unknown. The changes to the ScreenOS code replaced Juniper's chosen Q with one selected by the attacker.

From one perspective, the Juniper incident is just a particularly intricate software vulnerability, which is interesting on its own terms. More importantly, however, it sheds light on the contentious topic of "exceptional access" technologies which would allow law enforcement officials to gain access to the plaintext for encrypted data. A key component of any exceptional access system is restricting access to authorized personnel, with the most commonly proposed approach being encrypting the target keying material under a key (or keys) known to law enforcement which are then kept under tight control. The use of Dual EC in ScreenOS creates what is in effect an exceptional access system with Q as the public key and the discrete log of Q as the private decryption key. Historically, analysis of exceptional access systems has focused on the difficulty of controlling the decryption keys. In the specific case of ScreenOS, we do not know whether anyone had access to the corresponding key, but the Juniper incident starkly illustrates another risk: that of an attacker modifying a system's exceptional access capability in order to replace the authorized public key with one under her control, thus turning an exceptional access system designed for use by law enforcement into one which works for the attacker.

In this paper, we attempt to tell the story of that incident, pieced together by forensic reverse engineering of dozens of ScreenOS firmware revisions stretching back nearly a decade, as well as experimental validation on NetScreen hardware. We first provide background on Dual EC itself, then examine the way that it is used in ScreenOS and why this leads to such a severe vulnerability, then

The original version of this paper is entitled "A Systematic Analysis of the Juniper Dual EC Incident" and was published in *Proceedings of the 23rd ACM Conference on Computer and Communications Security* (Vienna, 2016), 468–479.

move to examine the history of the incident itself, and finally consider what lessons we can draw from this story.

2. DUAL EC IN SCREENOS

Cryptographic systems typically include deterministic PRNGs that expand a small amount of secret internal state into a stream of values which are intended to be indistinguishable from true randomness. An attacker able to predict the output of a PRNG will often be able to break any protocol implementation dependent on it, for instance by being able to predict cryptographic keys (which should remain secret) or nonces (which should often remain unpredictable).

Dual EC is a cryptographic PRNG standardized by National Institute of Standards and Technology (NIST) which is based on operations on an elliptic curve. Dual EC has three public parameters: the elliptic curve and two points on the curve called P and Q . ScreenOS uses the elliptic curve P-256 and sets P to be P-256's standard generator as specified in NIST Special Publication 800-90.⁴ That standard also specifies the Q to use, but ScreenOS uses Juniper's own elliptic curve point Q instead. The finite field over which P-256 is defined has roughly 2^{256} elements. Points on P-256 consist of pairs of 256-bit numbers (x, y) that satisfy the elliptic curve equation. The internal state of Dual EC is a single 256-bit number s .

Let $x(\cdot)$ be the function that returns the x -coordinate of an elliptic curve point; $\parallel$ be concatenation; $\text{lsb}_n(\cdot)$ be the function that returns the least-significant n bytes of its input in big-endian order; and $\text{msb}_n(\cdot)$ be the function that returns the most-significant n bytes. Starting with an initial state s_0 , one invocation of Dual EC implementation generates a 32 pseudorandom byte *output* and a new state s_2 as

$$\begin{aligned} s_1 &= x(s_0 P) & r_1 &= x(s_1 Q) \\ s_2 &= x(s_1 P) & r_2 &= x(s_2 Q) \\ \text{output} &= \text{lsb}_{30}(r_1) \parallel \text{msb}_2(\text{lsb}_{30}(r_2)), \end{aligned}$$

where sP and sQ denote scalar multiplication on P-256.

In 2007, Shumow and Ferguson showed¹⁶ that Dual EC was subject to a state reconstruction attack by an adversary who knows the value d such that $P = dQ$ and who can observe a single output value. The key insight is that multiplying the point $s_1 Q$ by d yields the internal state $x(d \cdot s_1 Q) = x(s_1 P) = s_2$. Although $s_1 Q$ is itself not known, 30 of the 32B of its x -coordinate (namely r_1) constitute the first 30B of *output*, and the attacker can guess the remaining bytes; the x -coordinate of an elliptic curve point determines its y -coordinate up to sign.

Assuming that the attacker knows the discrete log of Q , the major difficulty is recovering a complete output value; an attacker who only knows part of the value must exhaustively search the rest. The number of candidates grows exponentially as fewer bytes of r_1 are revealed, and recovery is intractable with fewer than about 26B. In ScreenOS, Dual EC is always used to generate 32B of output at a time, and therefore the attack is straightforward. When 30B of r_1 are

available, as in Juniper's implementation, the attacker must consider 2^{16} candidate points. From the attacker's perspective, this is the optimal situation.

Importantly, as far as is publicly known, Dual EC is secure against an attacker who knows P and Q but does not know d , as recovering d would require the ability to compute discrete logarithms, which would break elliptic curve cryptography in general.

3. THE SCREENOS PRNG SUBSYSTEM

Listing 1 shows the decompiled source code for the functions implementing the PRNG in ScreenOS version 6.2.0r1; the same function is present in other releases in the 6.2 and 6.3 series. It consists of two PRNGs, Dual EC and ANS X9.31 (Appendix A.2.4; Ref.²).

Note that identifiers such as function and variable names are not present in the binary; we assigned these names based on our analysis of the apparent function of each symbol. Similarly, specific control flow constructs are not preserved by the compilation/decompilation process. For instance, the `for` loop on line 21 may in fact be a `while` loop or some other construct in Juniper's source code. Decompilation does, however, preserve the functionality of the original code. For clarity, we have omitted Federal Information Processing Standards (FIPS) checks that ensure that the X9.31 generator has not generated duplicate output.

A superficial reading of the `prng_generate()` function suggests that Dual EC is used only to generate keys for the X9.31 PRNG, and that it is the output of X9.31 that is returned to callers (in the output global buffer). The Dual EC vulnerability described in Section 2 requires raw Dual EC output, so it cannot be applied. Indeed, a 2013 knowledge base article by Juniper⁸ claims exactly this. (We discuss this knowledge base article further in Section 6.)

Listing 1: The core ScreenOS 6.2 PRNG subroutines.

```

1 char block[8], seed[8], key[24]; // X9.31 vars
2 char output[32]; // prng_generate output
3 unsigned int index, calls_since_reseed;
4
5 void prng_reseed(void) {
6     calls_since_reseed = 0;
7     if (dualec_generate(output, 32) != 32)
8         error("[...] unable to reseed\n", 11);
9     memcpy(seed, output, 8);
10    index = 8;
11    memcpy(key, &output[index], 24);
12    index = 32;
13 }
14
15 void prng_generate(void) {
16     int time[2] = { 0, get_cycles() };
17     index = 0;
18     ++calls_since_reseed;
19     if (!one_stage_rng())
20         prng_reseed();
21     for (; index <= 31; index += 8) {
22         // FIPS checks removed for clarity
23         x9_31_generate_block(time, seed, key, block);
24         // FIPS checks removed for clarity
25         memcpy(&output[index], block, 8);
26     }
27 }
```

In this reading, the `prng_reseed()` function is occasionally invoked to reseed the X9.31 PRNG state. This function invokes the Dual EC generator, directing its output to the 32B buffer output. From this buffer, it extracts a seed and cipher key for the X9.31 generator. With X9.31 seeded, the `prng_generate()` function generates 8B of X9.31 output at a time (line 23) into output, looping until it has generated 32B of output (lines 21–26). Each invocation of `x9_31_generate_block` updates the X9.31 seed state in the seed buffer.

The straightforward reading given above is wrong.

First, and most importantly, `index`, the control variable for the loop that invokes the X9.31 PRNG in `prng_generate()` at line 21, is a global variable. The `prng_reseed()` function, if called, sets it to 32, with the consequence that, whenever the PRNG is reseeded, `index` is already greater than 31 at the start of the loop and therefore no calls to the X9.31 PRNG are executed.^a

Second, in the default configuration, `one_stage_rng()` always returns false, so `prng_reseed()` is always called. In the default configuration, then, the X9.31 loop is *never* invoked. (There is an undocumented ScreenOS command, `set key one-stage-rng`, that makes `one_stage_rng()` always return true; running this command induces a different PRNG vulnerability, discussed in the full version of this paper.⁵)

Third, the `prng_reseed()` happens to use the output global buffer as a staging area for Dual EC output before it copies parts of that output to the other global buffers that hold the X9.31 seed and key. This is the same global buffer that the `prng_generate()` function was supposed to fill with X9.31 output, but fails to. When callers look for PRNG output in output, what they find is 32B of raw Dual EC output.

For comparison, Listing 2 shows the decompiled source code for the PRNG function in ScreenOS 6.1, before Juniper's revamp. In ScreenOS 6.1, the loop counter, `index`, is a local variable rather than a global; the X9.31 PRNG is reseeded from system entropy every 10,000 calls, instead of every call

```

1 char block[8], seed[8], key[24]; // X9.31 vars
2 unsigned int calls_since_reseed;
3
4 void prng_generate(char *output) {
5     unsigned int index = 0;
6     // FIPS checks removed for clarity
7     if (calls_since_reseed++ > 9999)
8         prng_reseed();
9     // FIPS checks removed for clarity
10    int time[2] = { 0, get_cycles() };
11    do {
12        // FIPS checks removed for clarity
13        x9_31_generate_block(time, seed, key, block);
14        // FIPS checks removed for clarity
15        memcpy(&output[index], block, min(20-index, 8));
16        index += min(20-index, 8);
17    } while (index <= 19);
18 }

```

^a The global variable reuse was first publicly noted by Willem Pinckaers on Twitter. Online: https://twitter.com/_dvorak/status/679109591708205056, retrieved February 18, 2016.

and from Dual EC; and PRNG output is placed in a caller-supplied buffer instead of a global variable.

In addition, the ScreenOS 6.1 PRNG subsystem produces 20B at a time, not 32B as in ScreenOS 6.2 and 6.3. We discuss the significance of this difference in the next section.

4. INTERACTION WITH IKE

ScreenOS implements the Internet Protocol Security (IPsec) VPN protocol. To choose the keys that protect a VPN session, the client and the ScreenOS device perform an Internet Key Exchange (IKE)^{7,11} handshake.

Listing 2: The core ScreenOS 6.1 PRNG subroutine.

In the same version 6.2 release of ScreenOS that added Dual EC (Section 2) and modified the PRNG subsystem to expose raw Dual EC output (Section 3), Juniper made a cluster of IKE implementation changes that make it possible for an attacker who knows the Dual EC secret d to decrypt VPN connections. In the remainder of these sections, we provide a brief description of the relevant features of IKE and then explain the impact of these changes.

4.1. Overview of IKE

IKE and its successor IKEv2 are traditional Diffie–Hellman-based handshake protocols in which two endpoints (dubbed the *initiator* and the *responder*) establish a Security Association (SA) consisting of parameters and a set of keys used for encrypting traffic. Somewhat unusually, IKE consists of two phases:

Phase 1 establishes an “IKE SA” that is tied to the endpoints but not to any particular class of non-IKE network traffic. In this phase, the two sides exchange Diffie–Hellman (DH) shares and nonces, which are combined to form the derived keys. The endpoints may be authenticated in a variety of ways including a signing key and a statically configured shared secret.

Phase 2 establishes SAs that protect non-IKE traffic (typically IPsec). The IKE messages for this phase are protected with keys established in the first phase. This phase may involve a DH exchange but may also just consist of an exchange of nonces, in which case the child SA keys are derived from the shared secret established in the first phase.

IKEv2 refers to these phases as “Initial Exchange” and “CREATE_CHILD_SA,” respectively; for simplicity we will use the IKEv1 Phase 1/Phase 2 terminology in the rest of this article.

An attack on IKE where ScreenOS is the responder would proceed as follows: (1) using the responder nonce in the first phase, compute the Dual EC state; (2) predict the responder's DH private key and use that to compute the DH shared secret for the IKE SA, which is used to generate the first set of keys; (3) using these traffic keys decrypt the second phase traffic to recover both initiator and responder nonces and public keys; (4) recover the responder's private key, either by running Dual EC forward (the best case scenario) or by repeating the Dual EC attack using the new responder nonce; (5) use the responder's private key and the initiator's public key to compute the shared secret for the second phase SA and

thereby the traffic keys; and (6) use the traffic keys to decrypt the VPN traffic.

However, while this is straightforward in principle, there are a number of practical complexities and potential implementation decisions which could make this attack easier or more difficult (or even impractical) as described below.

4.2. Nonce size

For Dual EC state reconstruction to be possible, the attacker needs more than just to see raw Dual EC output. She needs at least 26B of the x -coordinate of a single elliptic-curve point to recover the Dual EC state; fewer bytes would be insufficient (Section 2).

Luckily for the attacker, the first 30B of the 32B returned by ScreenOS's Dual EC implementation belong to the x -coordinate of a single point, as we saw in Section 2. Luckily again for the attacker, ScreenOS's PRNG subsystem also returns 32B when called, and these are the 32B returned by a Dual EC invocation, as we saw in Section 3. Finally, IKE nonces emitted by ScreenOS are 32B long and produced from a single PRNG invocation. To summarize: In ScreenOS 6.2 and 6.3, IKE nonces always consist of 30B of one point's x -coordinate and 2B of the next point's x -coordinate—the best-case scenario for Shumow–Ferguson reconstruction.

It is worth expanding on this point. The IKE standards allow any nonce length between 8 and 256B (Section 5; Ref. 7). An Internet-wide scan of IKE responders by Adrian et al.³ found that a majority use 20B nonces. We are not aware of any cryptographic advantage to nonces longer than 20B. ScreenOS 6.1 sent 20B nonces and, as we noted in Section 3, its PRNG subsystem generated 20B per invocation. In ScreenOS 6.2, Juniper introduced Dual EC, rewrote the PRNG subsystem to produce 32B at a time, and modified the IKE subsystem to send 32B nonces.

4.3. NONCES AND DH KEYS

An attacker who knows the d corresponding to Juniper's point Q and observes an IKE nonce generated by a ScreenOS device can recompute the device's Dual EC state at nonce generation time. She can roll that state forward to predict subsequent PRNG outputs, though not back to recover earlier outputs. ScreenOS uses its PRNG to generate IKE Diffie–Hellman shares, so the attacker will be able to predict DH private keys generated after the nonce she saw and compute the session keys for the VPN connections established using those IKE handshakes.

This scenario is clearly applicable when the attacker has a network tap close to the ScreenOS device, and can observe many IKE handshakes. But what if the attacker's network tap is close to the VPN *client* instead? She might observe only a single VPN connection. If the nonce for a connection is generated after the DH share, the attacker will not be able to recover that session's keys.

A superficial reading of the ScreenOS IKE code seems to rule out single-connection attacks: The KE payload containing the DH share is indeed encoded *before* the Nr payload containing the nonce.

Conveniently for the attacker, however, ScreenOS also

contains a pre-generation feature that maintains a pool of nonces and DH keys that can be used in new IKE connections, reducing handshake latency. The pooling mechanism is quite intricate and appears to be designed to ensure that enough keys are always available while avoiding consuming too much run time on the device.

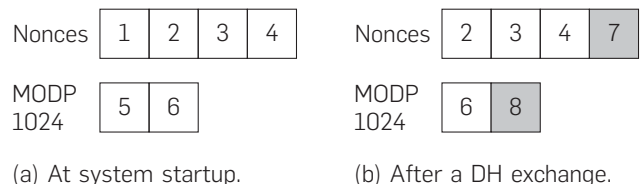
Independent First In, First Out (FIFO) queues are maintained for nonces, for each supported finite field DH group (MODP 768, MODP 1024, MODP 1536, and MODP 2048), and (in version 6.3) for each supported elliptic curve group (ECP 256 and ECP 384). The sizes of these queues depend on the number of VPN configurations that have been enabled for any given group. For instance, if a single configuration is enabled for a group then that group will have queue size of 2. The size of the nonce queue is set to be twice the aggregate size of all of the DH queues. At startup, the system fills all queues to capacity. A background task that runs once per second adds one entry to a queue that is not full. If a nonce or a DH share is ever needed when the corresponding queue is empty, a fresh value is generated on the fly.

The queues are filled in priority order. Crucially, the nonce queue is assigned the highest priority; it is followed by the groups in descending order of cryptographic strength (ECP 384 down to MODP 768). This means that in many (but not all) cases, the nonce for an IKE handshake will have been drawn from the Dual EC output stream *earlier* than the DH share for that handshake, making single-connection attacks feasible.

Figure 1 shows a (somewhat idealized) sequence of generated values, with the numbers denoting the order in which queue entries were generated, before and after an IKE Phase 1 exchange. Figure 1a shows the situation after startup: The first four values are used to fill the nonce queue and the next two values are used to generate the DH shares. Thus, when the exchange happens, it uses value 1 for the nonce and value 5 for the key, allowing the attacker to derive the Dual EC state from value 1 and then compute forward to find the DH share. After the Phase 1 exchange, which consumes a DH share and a nonce, and after execution of the periodic, queue-refill task, the state is as shown in Figure 1b, with the new values shaded.

Depending on configuration, the IKE Phase 2 exchange would consume either a nonce and a DH share or just a nonce. If the exchange uses both a nonce and a DH share, the dequeued nonce will again have been generated before

Figure 1. Nonce queue behavior during an IKE handshake. Numbers denote generation order, and values generated after the handshake are shaded. During a DH exchange, outputs 1 and 5 are used as the nonce and key, advancing the queue, and new outputs are generated to fill the end of the queue.



the dequeued DH share. That property will continue to hold for subsequent IKE handshakes, provided that handshakes do not entirely exhaust the queues. Had the refill task not prioritized refilling the nonce queue before any DH group queue, single-connection attacks would not have been possible. Had the nonce queue been the same length as a DH share queue, single-connection attacks would not have been possible in configurations where IKE Phase 2 consumed a nonce but not a DH share.

ScreenOS 6.1 pregenerates DH shares but not nonces; the nonce queues we have described were added in ScreenOS 6.2, along with Dual EC. Had nonce queues not been added, no handshakes would have been vulnerable to single-connection decryption attacks.

In the presence of multiple nonce-only Phase-2 exchanges within a single Phase-1 exchange, multiple DH groups actively used in connections, queue exhaustion, or certain race conditions, the situation is more complicated, and it is possible for an IKE handshake phase to have its DH share generated before its nonce. Single-connection decryption attacks would fail for those handshakes. Refer to the full version of this paper for details.⁵

4.4. Recovering traffic keys

If the attacker can predict the Diffie–Hellman private key corresponding to the ScreenOS device’s DH share for an IKE exchange, she can compute the DH shared secret for that exchange. With knowledge of the DH shared secret, computing the session keys used to encrypt and authenticate the VPN session being set up is straightforward, though the details depend on the IKE protocol version and the way in which the endpoints authenticate each other; for details, see the full version of this paper.⁵

For IKEv1 connections authenticated with digital signatures, the attacker knows everything she needs to compute the session keys. For IKEv1 connections authenticated with public key encryption, each peer’s nonce is encrypted under the other’s Rivest–Shamir–Adleman (RSA) public key, stopping the attack. IKEv1 connections authenticated with preshared keys fall somewhere in the middle: The attacker will need to know the preshared key in addition to the DH shared secret to compute the session keys. If the preshared key is strong, then the connection will still be secure. Fortunately for the attacker, many real-world VPN configurations use weak preshared keys (really passwords); in such cases having recorded an IKE handshake and recovered the DH shared secret, the attacker will be able to mount an offline dictionary attack on the preshared key. By contrast, the attacker will be able to compute session keys for IKEv2 connections in the same way, regardless of how they are authenticated.

Having computed the session keys, the attacker can decrypt and read the VPN traffic and, if she wishes, can tamper with it.

5. EXPERIMENTAL VALIDATION

To validate the attacks we describe above, we purchased a Juniper Secure Services Gateway 550M VPN device. We generated our own point Q and corresponding Dual EC secret d .

We modified firmware version 6.3.0r12 to put in place our point Q , matching Dual EC Known Answer Test (KAT) values, and the (non-cryptographic) firmware checksum, and we installed the modified firmware on our device. (Our device did not have a code-signing certificate installed, so we did not need to create a valid cryptographic signature for our modified firmware.)

Using the new firmware, we configured the device with three separate VPN gateways, configured for IKEv1 with a preshared key, IKEv1 with a 1024-bit RSA signing certificate, and IKEv2 with a preshared key, respectively. We made connections to each gateway using the strongSwan VPN software as our initiator and recorded the traffic to our device. We successfully decrypted each connection by recovering the Dual EC state and traffic keys using just that connection’s captured packets.

6. HISTORY OF THE JUNIPER INCIDENT

The history of the Juniper incident begins nearly a decade ago.^b In October 2008, Juniper released ScreenOS 6.2. As described in detail above, this release (1) replaced an entropy-gathering procedure for (re)seeding the ANS X9.31 PRNG with Dual EC using a custom Q point; (2) modified the X9.31 reseed logic to reseed on every call rather than every ten thousand calls; (3) changed the loop counter in the `prng_generate` procedure as well as the procedure’s output to be global variables, shared with the reseed procedure, thus ensuring that pseudorandom values are generated by Dual EC, and not X9.31; (4) changed the IKE nonce length from 20B to 32B; and (5) added a nonce pre-generation queue.

The result of the first four changes is that whoever knew the integer d corresponding to Juniper’s Q could passively decrypt (some) VPN traffic. Each of the first four changes is critical to the attack described in this article. The fifth change enables single-connection attacks in many cases, but is not necessary for multi-connection attacks.

This state of affairs continued for four years. At some point prior to the release of ScreenOS 6.2.0r15 (September 2012) and ScreenOS 6.3.0r12 (August 2012), someone modified Juniper’s source code. Based on the patched firmware revisions Juniper would later release, the modifications were quite small: The x -coordinate of Juniper’s Dual EC’s Q was changed as was the expected response to Dual EC’s Known Answer Test. As a result, the set of people who could passively decrypt ScreenOS’s VPN traffic changed from those who know Juniper’s d (if any) to those who know the new d corresponding to the changed Q (presumably the attacker who made the change).

Apparently unrelated to the 2012 changes, a second source code modification was made. A hard-coded SSH and Telnet password was inserted into Juniper’s code at some point before the release of ScreenOS 6.3.0r17 (April 2013). Logging in with this password yields administrator access.

^b The dates in this section come from file dates, ScreenOS release notes, and Juniper’s website, none of which agree precisely on *any* dates.

In early September 2013, the New York Times published an article based on documents from Snowden strongly implying that the National Security Agency (NSA) had engineered Dual EC to be susceptible to attack.¹⁵ The article does not name Dual EC; it instead refers to a 2006 NIST standard with a “fatal weakness, discovered by two Microsoft cryptographers in 2007,” presumably referring to Dan Shumow and Niels Ferguson’s presentation at CRYPTO 2007.¹⁶ This reporting led NIST to withdraw its recommendation for Dual EC.¹⁴

After NIST withdrew its recommendation, Juniper subsequently published a knowledge base article explaining their use of Dual EC in ScreenOS.

ScreenOS does make use of the Dual_EC_DRBG standard, but is designed to not use Dual_EC_DRBG as its primary random number generator. ScreenOS uses it in a way that should not be vulnerable to the possible issue that has been brought to light. Instead of using the NIST recommended curve points it uses self-generated basis points and then takes the output as an input to FIPS/ANSI X.9.31 [sic] PRNG, which is the random number generator used in ScreenOS cryptographic operations.⁸

The first mitigation—using self-generated basis points—only defends against the attacks described in this paper if Q is generated so that nobody knows d ; Juniper has provided no evidence that this is the case. As we describe in Section 3, Juniper’s claim that the output of Dual EC is only used as an input to X9.31 is incorrect.

This was the situation on December 17, 2015 when Juniper issued an out-of-cycle security bulletin⁹ for two security issues in ScreenOS: CVE-2015-7755^c (“Administrative Access”) and CVE-2015-7756^d (“VPN Decryption”).

This announcement was particularly interesting because it was not the usual report of developer error, but rather of malicious code which had been inserted into ScreenOS by an unknown attacker:

During a recent internal code review, Juniper discovered unauthorized code in ScreenOS that could allow a knowledgeable attacker to gain administrative access to NetScreen® devices and to decrypt VPN connections. Once we identified these vulnerabilities, we launched an investigation into the matter, and worked to develop and issue patched releases for the latest versions of ScreenOS.¹⁰

The bulletin prompted a flurry of reverse-engineering activity around the world, including by our team. The “Administrative Access” issue was quickly identified as the 2013 source code modification. This issue has been extensively discussed by Moore.¹³ Our analysis of the “VPN Decryption” issue, described in this article, shows that the 2012 code modification is responsible.

Our analysis implies several items of note. First, the 2012 code modification indicates that Juniper’s 2013 knowledge base article⁸ is incorrect when it states that

ScreenOS uses Juniper’s own Q point since, at that time, ScreenOS was shipping with the attacker’s Q . Second, by the end of 2015, Juniper knew that Dual EC could be exploited in ScreenOS. Despite this, Juniper’s initial fix was to revert the Q point to their initial value in each affected ScreenOS revision. Eventually, after press coverage of our results, Juniper committed to removing Dual EC from their PRNG subsystem.

7. EXCEPTIONAL ACCESS AND NOBUS

Law enforcement officials have been warning since 2014 that they are “going dark”: that ubiquitous end-to-end encryption threatens investigations by rendering intercepted communications unreadable. They have called on technology companies to rearchitect their products so intercepted communications could be decrypted given a court order. Computer scientists have resisted such “exceptional access” mandates, arguing that whatever mechanism implements it would constitute a vulnerability that might be exploited by third parties.¹

Attempts to design exceptional access mechanisms which do not introduce vulnerabilities go back at least as far as 1993, when the NSA introduced “Clipper,” an encryption algorithm embedded in a hardware platform with a built-in “key escrow” capability, in which cryptographic keys were separately encrypted under a key known to the US government. Such a mechanism would be “NOBUS,” in the jargon of the NSA, for “nobody but us” (p. 281; Ref.⁶): data would be cryptographically secure against anyone who did not have the keys but transparent to those who did.

While the key escrow mechanism designed for Clipper involved encrypting the traffic keys under the escrow key, it is also possible to build an exceptional access mechanism around a system like Dual EC, with the escrow key being the discrete log of Q . The common thread here is that the *key* is intended to be known only to authorized personnel.

Whatever the intent of Juniper’s selection of Dual EC, its use created what was in effect an exceptional access system: one where the key was the d value corresponding to Juniper’s choice of Q . We have no way of knowing whether anyone knew that d value or not, and Juniper has not described how they generated Q . However, around 2012, some organization gained the ability to make changes to Juniper’s source code repository. They used that access to change the Dual EC point Q to one of their choosing, in essence swapping out the escrow key. Between September 2012 and December 2015, official releases of ScreenOS distributed by Juniper included the intruders’ point Q instead of Juniper’s. VPN connections to NetScreen devices running affected releases were subject to decryption by the intruders, assuming they know the d corresponding to their point Q .

8. LESSONS

The ScreenOS vulnerabilities we have studied provide important broader lessons for the design of cryptographic systems, which we summarize here.

^c Of course, reducing nonce size cannot prevent all data exfiltration strategies. However, it may increase the difficulty of hiding the necessary code, and the complexity of executing an attack.

^c <https://cve.mitre.org/cgi-bin/cvename.cgi?name=cve-2015-7755>

^d <https://cve.mitre.org/cgi-bin/cvename.cgi?name=cve-2015-7756>

8.1. For protocol designers

Allowing nonces to vary in length, and in particular to be larger than necessary for uniquely identifying sessions, may be a bad idea. The authors are unaware of any cryptographic rationale for 256B nonces, as permitted by IPsec; it is simply an invitation for implementations to disclose sensitive state, intentionally or not.^c

Adding even low-entropy shared secrets as key derivation inputs helps protect against entropy failures. We observe a difference in exploitability of the ScreenOS bugs between IKEv1 and IKEv2 that is entirely due to the different use of the preshared key between the two protocols. It is unfortunate that IKEv2 is easier to exploit.

8.2. For implementers and code reviewers

Cryptographic code must be locally auditable: It must be written in such a way that examining a function or a module in isolation allows the reader to understand its behavior.

ScreenOS's implementation failed to live up to this guideline. A loop counter in the core `prng_generate` routine was defined as a global variable and changed in a subroutine. This is a surprising-enough pattern that several experienced researchers who knew that the routine likely had a bug failed to spot it before Willem Pinckaers' contribution. The `prng_generate` and `prng_reseed` routines reuse the same 32B buffer, `output`, for two entirely different purposes: Dual EC output with which to seed X9.31, and output from the PRNG subsystem. ScreenOS's use of pregeneration queues makes it difficult to determine whether nonces or Diffie-Hellman shares are generated first. Someone reading the code for the top-level functions implementing IKE in isolation will conclude that Diffie-Hellman shares are generated first, whereas in practice the opposite is usually the case.

The state recovery attacks suffered by Juniper suggest that implementations may wish to avoid revealing the raw output of a random number generator entirely, perhaps by hashing any PRNG output before using it as a nonce. One could also design implementations so that separate PRNGs are used for different protocol components, to separate nonce security from key security.

Several of the above mistakes represent poor software engineering practices. Cryptographic code reviews, whether internal or external (e.g., for FIPS validation), should take code quality into account.

8.3. For NIST

Juniper followed then-current best practices in designing and verifying their random number generators. They used a NIST-certified algorithm, followed the FIPS-recommended procedure to verify the output using test vectors, and followed a commonly recommended engineering guideline to use a PRNG as a whitener for a potentially insecure random number generator, removing—at least in theory—the structured output that makes Dual EC vulnerable.

In this case, all three approaches failed. In particular, a crippling defect in the whitening countermeasure managed to go undetected in FIPS certification. This suggests potential future work for research in the verification of

cryptographic systems. One step would be to track the origin and use of any buffers—especially shared buffers—and enforce a rule that all random number generator output can be traced back to an appropriate cryptographic function, such as a block cipher or hash. Some form of coverage analysis might also have revealed that the whitening is never performed.

To the extent that FIPS guidelines mandate the use of global state, they run counter to our suggestion, above, that cryptographic code be locally auditable.

Products are evaluated against FIPS standards by accredited laboratories. ScreenOS was FIPS certified with the X9.31 PRNG, yet the lab evaluating ScreenOS failed to spot that X9.31 was never invoked, as well as failing to detect the defect in the Dual EC implementation described in Section 3. NIST should revisit its laboratory accreditation program to ensure more thorough audits, especially of randomness subsystem code.

8.4. For attackers

The choice by the attacker to target the random number generation subsystem is instructive. Random number generators have long been discussed in theory as a target for kleptographic substitution attacks,¹⁸ but this incident tells us that the threat is more real than has been known in the academic literature.

From the perspective of an attacker, by far the most attractive feature of the ScreenOS PRNG attack is the ability to significantly undermine the security of ScreenOS *without* producing any externally detectable indication that would mark the ScreenOS devices as vulnerable. This is in contrast to previous well-known PRNG failures, which were externally observable, and, in the case of the Debian PRNG flaw,¹⁷ actually detected through observational testing. Indeed, the versions of ScreenOS containing an attacker-supplied parameter appear to have produced output that was cryptographically indistinguishable from the output of previous versions, thus preventing any testing or measurement from discovering the issue.

8.5. For journalists

Much of the coverage of the Juniper disclosure has focused on the unauthorized changes made in 2012 to the randomness subsystem and in 2013 to the login code. By contrast, our forensic investigation of ScreenOS releases highlights the changes made in the 6.2 series, in 2008, as the most consequential.

These changes, which introduced Dual EC and changed other subsystems in such a way that an attacker who knew the discrete log of Q could exploit it, were, as far as we know, added by Juniper engineers, not by attackers. This raises a number of questions:

How was the new randomness subsystem for the ScreenOS 6.2 series developed? What requirements did it fulfill? How did Juniper settle on Dual EC? What organizations did it consult? How was Juniper's point Q generated?

^c Online: <https://oversight.house.gov/hearing/federal-cybersecurity-detection-response-and-mitigation/>.

We are not able to answer these questions with access to firmware alone. Juniper's source code version-control system, their bug-tracking system, their internal e-mail archives, and the recollections of Juniper engineers may help answer them.

Despite numerous opportunities, including public questions put to their Chief Security Officer and a congressional hearing on this incident,^f Juniper has either failed or explicitly refused to provide any further details.

8.6. For policymakers

Much of the debate about exceptional access has focused on whether it is possible to construct secure exceptional access mechanisms, where “secure” is defined as only allowing authorized access—presumably by law enforcement. It is readily apparent that one of the major difficulties in building such a system is the risk of compromise of whatever keying material is needed to decrypt the targeted data.

The unauthorized change to ScreenOS's Dual EC constants made in 2012 illustrates a new threat: the ability for another party to modify the target software to subvert an exceptional access mechanism for its own purposes, with only minimally detectable changes. Importantly, because the output of the PRNG appears random to any entity that does not know the discrete log of Q , such a change is invisible both to users and to any testing which the vendor might do. By contrast, an attacker who wants to introduce an exceptional access mechanism into a program which does not already have one must generally make a series of extremely invasive changes, thus increasing the risk of detection.

In the case of ScreenOS, an attacker was able to subvert a major product—one which is used by the federal government—and remain undiscovered for years. This represents a serious challenge to the proposition that it is possible to build an exceptional access system that is available only to the proper authorities; any new proposal for such a system should bear the burden of proof of showing that it cannot be subverted in the way that ScreenOS was.

Acknowledgments

This material is based in part upon work supported by the U.S. National Science Foundation under awards EFMA-1441209, CNS-1505799, CNS-1010928, CNS-1408734, and CNS-1410031; The Mozilla Foundation; a gift from Cisco; and the Office of Naval Research under contract N00014-14-1-0333.

References

1. Abelson, H., Anderson, R., Bellare, S.M., Benaloh, J., Blaze, M., Diffie, W., Gilmore, J., Green, M., Landau, S., Neumann, P.G., Rivest, R.L., Schiller, J.I., Schneier, B., Specter, M., Weitzner, D.J. Keys under doormats: Mandating insecurity by requiring government access to all data and communications. *Commun. ACM* 58, 10 (Oct. 2015), 24–26.
2. Accredited Standards Committee (ASC) X9, Financial Services. *ANS X9.31-1998: Digital signatures using reversible algorithms for the financial services industry (rDSA)*, 1998. Withdrawn.
3. Adrian, D., Bhargavan, K., Durumeric, Z., Gaudry, P., Green, M., Halderman, J.A., Heninger, N., Springall, D., Thomé, E., Valenta, L., VanderSloot, B., Wustrow, E., Zanella-Béguelin, S., Zimmermann, P. Imperfect forward secrecy: How Diffie-Hellman fails in practice. In

- Proceedings of CCS 2015*. C. Kruegel and N. Li, eds. ACM Press, New York, NY, Oct. 2015, 5–17.
4. Barker, E., Kelsey, J. NIST Special Publication 800-90: Recommendation for Random Number Generation Using Deterministic Random Bit Generators. Technical report, National Institute of Standards and Technology, June 2006.
 5. Checkoway, S., Maskiewicz, J., Garman, C., Fried, J., Cohnsey, S., Green, M., Heninger, N., Weinmann, R.-P., Rescorla, E., Shacham, H. A systematic analysis of the Juniper Dual EC incident. In *Proceedings of CCS 2016*. S. Halevi, C. Kruegel, and A. Myers, eds. ACM Press, New York, NY, Oct. 2016, 468–479.
 6. Granick, J.S. *American Spies: Modern Surveillance, Why You Should Care, and What To Do About It*. Cambridge University Press, Cambridge, 2017.
 7. Harkins, D., Carrel, D. The Internet Key Exchange (IKE). RFC 2409 (Proposed Standard), Nov. 1998. Obsolete by RFC 4306, updated by RFC 4109. Online: <https://tools.ietf.org/html/rfc2409>.
 8. Juniper Networks. Juniper Networks product information about Dual_EC_DRBG. Knowledge Base Article KB28205, Oct. 2013. Online: <https://web.archive.org/web/20151219210530/https://kb.juniper.net/InfoCenter/index?page=content&id=KB28205&mv=print&act=LIST>.
 9. Juniper Networks. 2015-12 Out of Cycle Security Bulletin: ScreenOS: Multiple Security issues with ScreenOS (CVE-2015-7755, CVE-2015-7756), Dec. 2015.
 10. Juniper Networks. Important announcement about ScreenOS®. Online: <https://forums.juniper.net/t5/Security-Incident-Response/Important-Announcement-about-ScreenOS/ba-p/285554>, Dec. 2015.
 11. Kaufman, C. Internet Key Exchange (IKEv2) Protocol. RFC 4306 (Proposed Standard), Dec. 2005. Obsolete by RFC 5996, updated by RFC 5282. Online: <https://tools.ietf.org/html/rfc4306>.
 12. Kelsey, J. Dual EC in X9.82 and SP 800-90A. Presentation to NIST VCAT committee, May 2014. Slides online http://csrc.nist.gov/groups/ST/crypto-review/documents/dualec_in_X982_and_sp800-90.pdf.
 13. Moore, H.D. CVE-2015-7755: Juniper ScreenOS Authentication Backdoor. <https://community.rapid7.com/community/infosec/blog/2015/12/20/cve-2015-7755-juniper-screensos-authentication-backdoor>, Dec. 2015.
 14. National Institute of Standards and Technology. NIST opens draft Special Publication 800-90A, recommendation for random number generation using deterministic random bit generators for review and comment. http://csrc.nist.gov/publications/nistbul/itlbul2013_09_supplemental.pdf, Sept. 2013.
 15. Perlroth, N., Larson, J., Shane, S. N.S.A. able to foil basic safeguards of privacy on Web. *The New York Times*, Sep. 5 2013. Online: <http://www.nytimes.com/2013/09/06/us/nsa-foils-much-internet-encryption.html>.
 16. Shumow, D., Ferguson, N. On the possibility of a back door in the NIST SP800-90 Dual Ec Prng. Presented at the Crypto 2007 rump session, Aug. 2007. Slides online: <http://rump2007.cr.yt.to/15-shumow.pdf>.
 17. Yilek, S., Rescorla, E., Shacham, H., Enright, B., Savage, S. When private keys are public: Results from the 2008 Debian OpenSSL vulnerability. In *Proceedings of IMC 2009*. A. Feldmann and L. Mathy, eds. ACM Press, New York, NY, Nov. 2009, 15–27.
 18. Young, A., Yung, M. Kleptography: Using cryptography against cryptography. In *Proceedings of Eurocrypt 1997*. W. Fumy, ed. volume 1233 of *LNCS*, Springer-Verlag, May 1997, 62–74.

Stephen Checkoway, University of Illinois at Chicago, IL, USA.

Jacob Maskiewicz, Eric Rescorla, Hovav Shacham, University of California, San Diego, CA, USA.

Christina Garman, Matthew Green, Johns Hopkins University, Baltimore, MD, USA.

Joshua Fried, Shaanan Cohnsey, Nadia Heninger, University of Pennsylvania, Philadelphia, PA, USA.

Ralf-Philipp Weinmann, Comsecuris, Duisberg, Germany.