A Systematic Analysis of the Juniper Dual EC Incident

Stephen Checkoway[*], Jacob Maskiewicz[†], Christina Garman[‡], Joshua Fried[§], Shaanan Cohney[§], Matthew Green[‡], Nadia Heninger[§], Ralf-Philipp Weinmann[¶], Eric Rescorla[†], Hovav Shacham[†]

[*]University of Illinois at Chicago, [†]University of California, San Diego, [‡]Johns Hopkins University, [§]University of Pennsylvania, [¶]Comsecuris

ABSTRACT

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

In this paper, we describe 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 since 2008 to passive exploitation by an attacker who selects the Dual EC curve point. This vulnerability arises due to apparent 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. We investigate the possibility of passively fingerprinting ScreenOS implementations in the wild. This incident is an important example of how guidelines for random number generation, engineering, and validation can fail in practice.

1. INTRODUCTION

In his statement for the record before the Senate Armed Services Committee on February 9, 2016, James Clapper, the U.S. Director of National Intelligence, illustrated the "worldwide threat assessment of the U.S. intelligence community" with an example of vulnerable infrastructure:

> A major U.S. network equipment manufacturer acknowledged last December that someone repeatedly gained access to its network to change source code in order to make its products' default encryption breakable. The intruders also introduced a default password to enable undetected access to some target networks worldwide. [10]

The "network equipment manufacturer" was Juniper Networks; it had disclosed the two issues in a security bulletin on December 17, 2015 [23], and released patched versions of ScreenOS, the operating system powering the affected NetScreen devices.

Immediately following Juniper's advisory, security researchers around the world — including our team — began examining the ScreenOS firmware to find the vulnerabilities Juniper claimed to have patched. They found that the change that, per Clapper, rendered ScreenOS encryption "breakable" did nothing but replace a few embedded constants.

In this paper, we explain how these changed constants may have allowed whoever introduced them to decrypt passively recorded VPN traffic to affected devices. The 2012 change took advantage of Juniper's 2008 overhaul of the ScreenOS randomness which introduced the NSA-designed Dual EC random number generator, and included Juniper-selected constants which we are unable to verify are secure. Juniper's December 2015 patch restored these original constants.

Our methods include forensic reverse engineering of dozens of ScreenOS firmware revisions stretching back nearly a decade; experimental testing on NetScreen hardware; and Internet measurement studies.

Juniper's NetScreen devices were FIPS certified, and the affected code implemented the standard IPsec protocol suite. Our findings thus have implications for many of the stakeholders in the development of cryptographic products, including protocol designers, implementers, code reviewers, and policymakers.

Pseudorandom number generators. Random number generation is critical to the implementation of cryptographic systems. Random numbers are used for a variety of purposes, including generation of nonces and cryptographic keys. Because generating a sufficient quantity of true random numbers via physical means is hard, cryptographic systems typically include deterministic *pseudorandom number generators* (PRNGs) which expand a small amount of secret internal state into a stream of values which are intended to be indistinguishable from true randomness.

Historically, random number generators have been a major source of vulnerabilities [8, 19, 22, 28, 45]. This is because an attacker who is able to predict the output of a PRNG will often be able to break any protocol implementation dependent on it. For instance, they may be able to predict any cryptographic keys (which should remain secret) or nonces (which should often remain unpredictable). Past PRNG failures have resulted from a failure to seed with sufficiently random data [19, 22] or from algorithms which are not *secure*, in

CCS '16 October 24–28, 2016, Vienna, Austria

ISBN 978-1-4503-4139-4/16/10. . . $15.00
DOI: http://dx.doi.org/10.1145/2976749.2978395

the sense that they allow attackers to recover the internal state of the algorithm from some public output.

The NSA-designed Dual EC PRNG [4, 27] 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. The changed constants in ScreenOS defined this Dual EC point Q.

Summary of our findings. Our analysis shows that from 2008 until it was patched in April 2016, Juniper's ScreenOS PRNG implementation was vulnerable to efficient state recovery attacks conducted by an attacker who could select the Q value. This capability is not an inevitable result of the known attacks on Dual EC, but instead stems from a collection of design choices made by Juniper in 2008.

We identified a constellation of changes made to both the PRNG and IKE implementations between ScreenOS 6.1 and 6.2 that together substantially predispose the IKE/IPsec implementation to state recovery attacks on the Dual EC generator. These changes, which were introduced concurrently with the addition of Dual EC, create a "perfect storm" of vulnerabilities that combine to enable a highly effective *single-handshake* exploit against the ScreenOS IKE implementation. Moreover, we identify several implementation decisions that superficially appear to reduce exploitability, but that on closer examination actually facilitate the attack. It is these changes — crucially, *not* the "unauthorized code" introduced by the third party — that enable passive VPN decryption for an attacker who has the discrete log of Q.

To validate the accuracy of our findings, we implement a proof of concept exploit against our SSG-550M running ScreenOS 6.3.0 and show that when the device is configured with a Q parameter of our choosing, our attacks can efficiently decrypt VPN connections from a single handshake, without seeing any other traffic. Moreover, we investigate the impact of different IPsec versions and configurations on the attack, and show that configuration decisions can substantially affect the exploitability of the device — in some cases rendering the device entirely secure.

2. Dual EC BACKGROUND

In this section, we describe the Dual EC pseudorandom number generator along with some details on how ScreenOS implements Dual EC. We also describe the attack on Dual EC described by Shumow and Ferguson [40],

Dual EC comes in a variety of forms. There are two slightly different NIST standards for Dual EC, which also contain optional features. There are three standard elliptic curves which can be used, and implementors are free to make a number of software engineering choices. Each of these design decisions can affect the difficulty of the Shumow–Ferguson attack. For concreteness, we describe Dual EC as implemented in Juniper's ScreenOS below. For more details on other forms of Dual EC, see Checkoway et al. [9].

Dual EC has three public parameters: the elliptic curve and two distinct 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 800-90A [34]. That standard also specifies the Q to use, but ScreenOS uses Juniper's own elliptic curve point 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.

In ScreenOS, Dual EC is always used to generate 32 bytes of output at a time. Let $x(\cdot)$ be the function that returns the x-coordinate of an elliptic curve point; $\|$ be concatenation; $\mathrm{lsb}_n(\cdot)$ be the function that returns the least-significant n bytes of its input in big-endian order; and $\mathrm{msb}_n(\cdot)$ be the function that returns the most-significant n bytes. Starting with an initial state s_0, Dual EC generates 32 pseudorandom bytes *output* and a new state s_2 as follows,

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

where sP and sQ denote scalar multiplication.

In 2007, Shumow and Ferguson [40] noted that if the elliptic-curve discrete logarithm $e = \log_P Q$ (i.e., the integer e such that $eP = Q$) were known, then seeing *output* would reveal the Dual EC internal state. The key insight is that one can obtain $d = \log_Q P = e^{-1} \bmod n$, where n is the group order, and then 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 32 bytes of its x-coordinate (namely r_1) is the first 30-bytes of *output*.

This insight gives rise to the simple procedure to recover s_2. For each of the 2^{16} 256-bit integers r such that $\mathrm{lsb}_{30}(r)$ equals the first 30-bytes of *output*, check if r is a valid x-coordinate of a point on the curve.[1] In other words, find a point R such that $x(R) = r$. Roughly half of the r values will be valid x-coordinates.[2] For each such R, compute $s' = x(dR)$ and $r' = x(s'Q)$. If the correct $r = r_1$ is chosen, $\mathrm{msb}_2\big(\mathrm{lsb}_{30}(r')\big)$ will be equal to the last two bytes of *output* and $s' = s_2$, the new internal state.

The one complication with the above procedure is that there may be several values of r such that $\mathrm{msb}_2\big(\mathrm{lsb}_{30}(r')\big) = \mathrm{lsb}_2(\textit{output})$ and each such r corresponds to a potential internal state s'. In practice, this is a minor complication as it is exceedingly rare for there to be more than three such r.

3. HISTORY OF THE JUNIPER INCIDENT

After NIST recommended against the use of Dual EC [34] in response to post-Snowden concerns about the default value of Q, Juniper published a knowledge base article [25] explaining their use of Dual EC in ScreenOS, the operating system powering its NetScreen firewall appliances, stating that although those products used Dual EC:

> ScreenOS does make use of the Dual_EC_DRBG standard, but is designed to not use Dual_EC_DBRG 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.

The first of these mitigations — self-generated basis points[3] — is not completely satisfactory because it depends on Juniper generating Q in such a way that nobody knows its discrete log, which they have not verifiably demonstrated. However, the second mitigation —

[1] This procedure is sometimes called point decompression.

[2] Each r that is an x-coordinate of some point R is also an x-coordinate of the point $-R$. It doesn't matter which point is chosen as R and $-R$ differ only in the "sign" of their y-component.

[3] While the Juniper article says "points", actually only Q differs from the NIST default values. Juniper's implementation uses the default P value.

if implemented correctly — defends against the current publicly known attacks on Dual EC because those attacks rely on having Dual EC output rather than a one-way function of that output, so even an attacker who knew the discrete log of Q would be unable to recover the PRNG state.

This was the situation on December 17, 2015 when Juniper issued an out-of-cycle security bulletin [23] for two security issues in ScreenOS:

- CVE-2015-7755[4] ("Administrative Access")
- CVE-2015-7756[5] ("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.

The "Administrative Access" vulnerability was determined to be a back door in the SSH [46] daemon that would have allowed anyone who knew the correct password to log in with administrative access. This issue has been extensively discussed by Moore [33]. The second issue, however, turns out to be far more technically interesting. According to Juniper's advisory:

> VPN Decryption (CVE-2015-7756) may allow a knowledgeable attacker who can monitor VPN traffic to decrypt that traffic. It is independent of the first issue.
>
> This issue affects ScreenOS 6.2.0r15 through 6.2.0r18 and 6.3.0r12 through 6.3.0r20. No other Juniper products or versions of ScreenOS are affected by this issue.
>
> There is no way to detect that this vulnerability was exploited.

While both Juniper's advisory and the CVE itself are short on details, comparison of the binaries for the vulnerable and patched versions reveal that the relevant change to the code is a change in the value of Q and the corresponding test vectors [43]. The natural inference, therefore, is that the attacker changed Q away from Juniper's original version (which is itself different from the default Q in the standard) and that the patched version changes it back. What makes this even more interesting is that — as noted above — even a Q value for which the attacker knows the discrete log should not lead to a passive decryption vulnerability because the output is supposed to be filtered through the ANSI X9.31 PRNG. This obviously raises serious questions about the accuracy of Juniper's 2013 description of their system, specifically:

1. Why does a change in Q result in a passive VPN decryption vulnerability?
2. Why doesn't Juniper's use of X9.31 protect their system against compromise of Q?
3. What is the history of the PRNG code in ScreenOS?
4. How was Juniper's Q value generated?
5. Is the version of ScreenOS with Juniper's authorized Q vulnerable to attack?

[4]http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2015-7755
[5]http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2015-7756

Table 1: ScreenOS firmware versions we examined.

Device series	Architecture	Version	Revisions
SSG-500	Intel x86	6.3.0	12b
SSG-5/SSG-20	ARM-BE	5.4.0	1–3, 3a, 4-16
		6.0.0	1–5, 5a, 6–8, 8a
		6.1.0	1–7
		6.2.0	1–8, 19
		6.3.0	1–6

Juniper's public statements regarding this incident set up a mystery. We have a body of material — an archive of firmware releases — that we use to answer many of the questions that make up that mystery. Specifically, we answer why changing Q matters and why X9.31 does not defend against that change in Section 4 and 5. By examining multiple versions of ScreenOS, we reconstruct a rough timeline of the evolution of ScreenOS's PRNG code in Section 9. Then there are some lingering questions, especially ones to do with motive or the identity of actors, that would require a different body of material to approach. We call these out in Section 9.1, and suggest specific resources that journalists might wish to consult to answer them.

4. THE SCREENOS PRNG

In this section, we describe the methodology and results of our analysis of the ScreenOS 6.2 PRNG cascade subroutines.[6]

Immediately following Juniper's 2015 announcement [23], security researchers around the world — including our team — began examining the ScreenOS firmware in an attempt to find the vulnerabilities Juniper claimed to have patched. HD Moore quickly posted a diff of the strings in the firmware between versions 6.2.0r14.0 and 6.2.0r15.0 — the version where the unauthorized code was introduced.[7] Line 934 shows a changed 32-byte hexadecimal constant. Based on the following hexadecimal constants, we hypothesized that this was the x-coordinate of an elliptic curve point on NIST curve P-256. By reverse engineering a copy of the firmware, it soon became apparent to our team and others that the changed constant was the x-coordinate of a nonstandard point Q in Dual EC.[8] Knowing that a third party changed the point Q in Dual EC is suggestive — but not dispositive — of that being key to being able to decrypt VPN connections described in Juniper's announcement. Understanding both how this could be exploited as well as when and how ScreenOS started using Dual EC required a closer, forensic investigation.

Towards those ends, our team acquired several Juniper Secure Services Gateway (SSG) devices running ScreenOS, including an Intel x86–based SSG-550M and an ARM-based SSG-5. Additionally, we acquired ScreenOS firmware binaries for 50 different point releases of ScreenOS across the two architectures and five major versions. Table 1 summarizes the versions we examined. We verified the MD5 or SHA-1 hash of each firmware version against those published on Juniper's website — except for the 6.0 revisions for which we could find no published hashes.

Each revision of the firmware contains a modified copy of the "engine" variant of OpenSSL 0.9.6c. Identifying OpenSSL's functions in a binary is particularly easy. One need only look up error and function ordinals used as arguments to error handling functions in lists of error codes included in the OpenSSL source code. Once the

[6]ScreenOS 6.3's cascade is identical.
[7]https://gist.github.com/hdm/0fbaf7408a6c7e0566c5
[8]Adam Langley documented some of this effort as it appeared on Twitter https://www.imperialviolet.org/2015/12/19/juniper.html.

Listing 1: The core ScreenOS 6.2 PRNG subroutines.

```
1  void prng_reseed(void) {
2    blocks_generated_since_reseed = 0;
3    if (dualec_generate(prng_temporary, 32) != 32)
4      error_handler("FIPS ERROR: PRNG failure, "
5                    "unable to reseed\n", 11);
6    memcpy(prng_seed, prng_temporary, 8);
7    prng_output_index = 8;
8    memcpy(prng_key,
9           &prng_temporary[prng_output_index], 24);
10   prng_output_index = 32;
11 }
12
13 void prng_generate(void) {
14   int time[2];
15   time[0] = 0;
16   time[1] = get_cycles();
17   prng_output_index = 0;
18   ++blocks_generated_since_reseed;
19   if (!one_stage_rng())
20     prng_reseed();
21   for (; prng_output_index <= 31;
22        prng_output_index += 8) {
23     // FIPS checks removed for clarity
24     x9_31_generate_block(time, prng_seed, prng_key,
25                          prng_block);
26     // FIPS checks removed for clarity
27     memcpy(&prng_temporary[prng_output_index],
28            prng_block, 8);
29   }
30 }
```

OpenSSL functions were identified, we identified the pseudorandom number generator functions and, ultimately, how those were used to construct nonces and Diffie–Hellman keys in IKE.

Extracting the strings from each of the firmware binaries revealed the presence of an OpenSSL elliptic curve component in versions 6.2.0r1 and later which suggested that that was where Dual EC was introduced. Nevertheless, we examined all of the five major versions for which we had firmware. For each of those major versions, we reverse-engineered the earliest revision we had, the latest revision we had, and one or more intermediate revisions to determine how the IKE nonce generation — including the pseudorandom number generator used — changed over time.

Listing 1 shows the decompiled source code for the ScreenOS PRNG version 6.2.0r1. Note that identifiers such as function and variable names are not present in the binary; we assigned these names based on 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 22 may in fact be a `while` loop or some other construct in the actual Juniper source. Decompilation does, however, preserve the functionality of the original code. For clarity, we have omitted FIPS checks that ensure that the ANSI X9.31 generator [1, Appendix A.2.4] has not generated duplicate output.

Superficially, the ScreenOS implementation appears consistent with Juniper's description: When `prng_generate()` is called, it first potentially reseeds the X9.31 PRNG state (lines 18 – 20) via `prng_reseed()`. When `prng_reseed()` is called, it invokes the Dual EC DBRG to fill the 32-byte buffer `prng_temporary`. From this buffer, it extracts a seed and cipher key for the ANSI X9.31 generator. Once the X9.31 PRNG state is seeded, the implementation then generates 8 bytes of X9.31 PRNG output at a time (line 25) into `prng_temporary`, looping until it has generated 32 bytes of output (lines 22–29), using the global variable `prng_seed` to store the ANSI X9.31 seed state, updating it with every invocation of `prng_generate_block()`.

However, upon closer inspection, the behavior of the generator is subtly different. This is due to two coupled issues: First, `prng_reseed()` and `prng_generate_blocks()` share the static buffer `prng_temporary`; and second, when `prng_reseed()` is invoked, it fills `prng_temporary` (line 3) and then sets the static variable `prng_output_index` to 32 (the size of the Dual EC output).[9] Unfortunately, `prng_output_index` is *also* the control variable for the loop that invokes the ANSI X9.31 PRNG in `prng_generate()` at line 22. The consequence is that whenever the PRNG is reseeded, `prng_output_index` is 32 at the start of the loop and therefore no calls to the ANSI X9.31 PRNG are executed. Thus, Dual EC output is emitted directly from the `prng_generate()` function.

In the default configuration, `one_stage_rng()` always returns false so X9.31 is reseeded on *every* call. There is an undocumented ScreenOS command, `set key one-stage-rng`, which is described by a string in the command-parsing data-structure as "Reduce PRNG to single stage." Invoking this command effectively disables reseeding until this setting is changed.

When combined with the cascade bug described above, disabling reseeding introduces a different security vulnerability: The first block emitted after reseed is precisely the concatenation of the 8-byte seed and the 24-byte key used for future blocks of output from the ANSI X9.31 PRNG. An attacker who is lucky enough to observe an immediate post-reseed output can predict the rest of the PRNG stream until the next reseed *even without knowing* $\log_P Q$.[10]

Had `prng_output_index` not been used in `prng_reseed`, the reuse of `prng_temporary` would be safe. As described in section 9, the index variable used in the for loop in `prng_generate` changed from a local variable to the `prng_output_index` global variable between the final version of ScreenOS 6.1 and the first version of 6.2.

5. INTERACTION WITH IKE

As suggested by the exploit description, the primary concern with a Dual EC implementation is that an attacker may be able to use public information emitted by the PRNG to extract the Dual EC internal state, and use this to predict future secret values. Because ScreenOS is not only a firewall but also a VPN device, the natural target is Internet Key Exchange (IKE) [21, 26], the key establishment protocol used for IPsec [30]. Note that the existence of a Dual EC generator does not by itself imply that Juniper's IKE implementation is itself exploitable, even in conditions where the attacker knows the Dual EC discrete log. There are a number of parameters that affect both the feasibility and cost of such an attack.

5.1 Overview of IKE

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

[9]The global variable reuse was first publicly noted by Willem Pinckaers on Twitter. Online: https://twitter.com/_dvorak_/status/679109591708205056, retrieved February 18, 2016.

[10]There are technical obstacles to overcome. X9.31 uses the current time (parameter DT in the specification; implemented as the processor cycle counter in ScreenOS) as an input to the PRNG. As long as the time value can be guessed (or brute forced), the X9.31 generator's output can be predicted. As `one-stage-rng` is off by default and the command that enables is is undocumented, we did not study this issue in depth.

- *Phase 1 (IKEv1)/Initial Exchange (IKEv2)*: Used to establish an "IKE SA" which is tied to the endpoints but not to any particular class of non-IKE network traffic.
- *Phase 2 (IKEv1)/CREATE_CHILD_SA (IKEv2)*: Used to establish SAs which protect non-IKE traffic (typically IPsec). The IKE messages for this phase are protected with keys established in the first phase. This phase may be run multiple times with the same phase 1 SA in order to establish multiple SAs (e.g., for different IP host/port pairs), but as a practical matter many VPN connections compute only one child SA and use it for all traffic.

For simplicity, we will use the IKEv1 terminology of phase 1/ phase 2 in the rest of this document.

IKE messages are composed of a series of "payloads" such as KE (key exchange), Ni (initiator nonce), and Nr (responder nonce).

The first IKE phase consists of a Diffie–Hellman exchange in which both sides exchange DH shares and a nonce, 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. The second IKE 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.

At this point, we have a conceptual overview of how to attack IKE where ScreenOS is the responder: (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; (3) using the keys derived from the IKE SA, decrypt the second phase traffic to recover the peers' nonces and public keys (in the best case, the responder nonce and private key can be computed by running Dual EC forward; otherwise one can repeat the Dual EC attack); and (4) use those to compute the shared secret for the second phase SA and thereby the traffic keys. Use those 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.

5.2 Nonce Size

The first question we must examine is whether the attacker ever gets a complete Dual EC block. As Checkoway et al. [9] describe in detail, it is only practical to exploit Dual EC if provided with nearly a complete point output. As specified, Dual EC emits only 30 bytes of the 32-byte point, which requires the attacker to try approximately half of the remaining 2^{16} values to find the state, and the work factor goes up exponentially with the number of missing bytes, so exploitation rapidly becomes impractical the less of the point the attacker has.

Many reasonable implementation strategies could result in an attacker obtaining only small fractions of a point. For example, unlike TLS, IKE has a variable-length nonce, which is required to be between 8 and 256 bytes in length [21, Section 5]. If a nonce length below 30 bytes were used, it could significantly increase the amount of work required to recover the Dual EC state

However, as of version 6.2 ScreenOS uses a 32-byte nonce made from two successive invocations of Dual EC, with the first supplying 30 bytes and the second supplying 2 bytes. As described in Section 2, this is nearly ideal from the perspective of the attacker because it can use the first 30 bytes (the majority of the point) to determine possible states, and then narrow the results by checking which states produce the correct value for the remaining two bytes. In practice, this usually results in 1 to 3 possible states.

5.3 Nonces and DH Keys

Although the IKE messages contain both a nonce and a DH share our analysis of Juniper's IKE implementation indicates that the KE payload containing the DH share is encoded *before* the Nr (nonce) payload. If (as is natural), the keys and nonces are generated in the same order as they are encoded, then it will not be possible to use Nr from one connection to attack that same connection. This is because Dual EC state recovery only allows you to predict future values, not recover past values. While not necessarily fatal to the attacker, because nonces generated in one connection might be used to predict the DH private keys generated in some subsequent connection; this would not be ideal from the attacker's perspective, especially if connection establishment is infrequent.

Conveniently for the attacker, however, ScreenOS also contains a pre-generation feature that maintains a pool of nonces and DH keys which can then be used in new IKE connections rather than generating them in the critical path (i.e., during the handshake). 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.

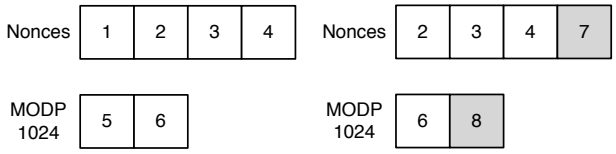
Summarized briefly, independent FIFO queues are maintained for nonces, each finite field DH group (MODP 768, MODP 1024, MODP 1536, and MODP 2048), and (in version 6.3) each elliptic curve group (ECP 256 and ECP 384). The sizes of these queues depend on the number of VPN configurations which have been 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 and disabled groups have a queue size of 0. The size of the nonce queue is set to be twice the aggregate size of all of the DH queues. So, for instance, if only the MODP 1024 group is configured, then the initial queue size will be (MODP 1024 = 2, nonce = 4). Or, if two VPN configurations are set to use MODP 1024 and one configuration is set to use MODP 1536, initial queue size will be (MODP 1024 = 4, MODP 1536 = 2, nonce = 12). At initial startup time, the system completely fills all the queues to capacity and then sets a timer that fires every second to refill the queues if any values have been used.[11] If a nonce or a DH key is ever requested when the queue is empty, then a fresh value is generated on the fly.

Importantly, the queues are filled in priority order with nonces being the highest priority 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 a given connection will precede the keys for that connection in the random number sequence.

Figure 1 shows a (somewhat idealized) sequence of generated values,[12] with the numbers indicating the order in which they were generated before and after an IKE DH 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

[11] Note: only one value is generated per second, so if several values are used, it takes some time to refill the queue.

[12] For simplicity, we represent multiple consecutive invocations of the PRNG as a single value and ignore invocations of the PRNG for non-IKE purposes. In addition, because the queues are refilled asynchronously with respect to the IKE exchanges, there is a race condition between values being consumed and being refreshed. The pattern shown here and below is the result of assuming that the timer fires between handshakes. If it fires more frequently (i.e., between each DH and nonce encoding), then the nonces become even and the DH shares become odd. Mixed patterns are also possible.



(a) At system startup. **(b)** After a DH exchange.

Figure 1: Nonce queue behavior during an IKE handshake. Numbers indicate 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.

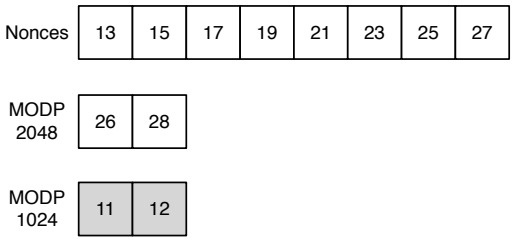


Figure 2: Queue state after 8 MODP 2048 exchanges. Numbers indicate generation order, and stale values are shaded. If several connections have been made to the same DH group, the other DH group can grow stale as all nonces that were generated before those keys are used up.

DH private key. After a single DH exchange, which requires one DH key and one nonce, the state is as shown in Figure 1b, with the new values shaded. Note that the next-in-line values continue to have the property that the nonce was generated before the DH share. Because nonce computation is prioritized over key generation, in this simple configuration where you have a single DH group that is used for every handshake, then as long as handshakes are done reasonably slowly (giving the background task enough time to fill the queue) the nonce used for a given handshake will always have been generated prior to the DH key for that handshake. Of course, if a large number exchanges are run in succession (i.e., outpacing the background task) it is possible to exhaust both queues entirely, at which point the request for a key or nonce will cause the value to be generated immediately, resulting in the DH key being computed before the nonce.

5.4 Non-DH Phase 2 Exchanges

As noted above, the phase 2 exchange need not include a new DH exchange; implementations can simply do a nonce exchange and generate fresh keys (although Juniper's documentation recommends doing DH for phase 2 as well [24, Page 72]). In this case, IKE will consume an additional nonce from the nonce queue but not a new DH key from the DH key queue. In the case where endpoints do a single phase 1 exchange and then a phase 2 exchange, with only the former doing DH, then setting up a VPN connection setup consumes two nonces and one DH key. However, because the nonce queue is twice as large as the DH queue, as long as the refill timer fires reasonably often with respect to the handshakes it is not possible to exhaust the nonce queue (thus causing a fresh PRNG value to be generated) while there is still a stale DH value in the DH queue. Note that if the nonce and DH queues were the same size, then non-DH phase 2 exchanges would frequently cause keys to be stale with respect to the nonce.

In addition, if *multiple* non-DH phase 2 exchanges are done within a single phase 1 exchange, then it is possible to empty the nonce queue while there are still values in the DH queue. In this case, it will only be possible to decrypt connections established using those values if the attacker has recorded previous nonces, rather than decrypting a connection in isolation. Similarly, the current nonces could be used to decrypt future connections but not the connections they are transmitted with.

5.5 Multiple Groups

If ScreenOS is configured to use multiple groups, then it is possible to have the shares for one group become stale with respect to the nonces queue, as shown in Figure 2, which shows the result of eight MODP 2048 exchanges on the queues. The shaded MODP 1024 values were all generated before any of the remaining nonces. If the attacker starts listening at this point and observes a MODP 1024 exchange, he will not be able to decrypt it.

5.6 Recovering traffic keys

As described above, IKE comes in two versions (IKEv1 and IKEv2) which are slightly different. Furthermore, each version uses a somewhat unusual two-phase approach to protecting traffic. In this section, we describe the phases and the authentication modes that determine whether or not protected traffic can be passively decrypted.

IKEv1, phase 1. IKEv1 defines four authentication modes for phase 1: digital signatures, two modes using public-key encryption, and preshared keys [21, Section 5]. Although the details vary, each mode computes a shared secret, SKEYID, derived from secret values (e.g., nonces and Diffie–Hellman keys) exchanged in the handshake. Next, the authentication keying material, SKEYID_a, and encryption keying material, SKEYID_e, are derived from SKEYID. Finally, the traffic keys used to protect phase 2 are derived from SKEYID_e in an algorithm-specific manner. We omit further discussion of authentication key material below as a passive adversary does not need it. An active adversary wishing to tamper with the traffic would need to use SKEYID_a in the straightforward manner prescribed by the RFC.

- **Authentication with digital signatures.** In this mode, the initiator and responder nonces and DH public keys are exchanged in the clear. Starting with the responder's nonce, an attacker who can recover the responder's DH private key has all of the material necessary to compute SKEYID and thus traffic keys.
- **Authentication with public key encryption.** IKEv1 defines two public-key encryption modes for authentication. The revised mode uses half the number of public-key encryptions and decryptions the other mode uses, but are otherwise similar. In these modes, the DH public keys are exchanged in the clear but each peer encrypts its nonce using the other's public key. These modes require the initiator to know the responder's public key prior to the handshake. Each peer decrypts the other's nonce and computes SKEYID. Since nonces are encrypted, even if an attacker can recover the responder's nonce (e.g., by capturing a nonce in the clear from a previous connection, recovering the Dual EC state, and walking the generator forward), the initiator's nonce is also encrypted, thus stopping the attack.
- **Authentication with preshared keys.** In this mode, a preshared key needs to be established out of band. The DH public keys and nonces are exchanged in the clear. The encryption keying material, SKEYID_e is derived from the preshared key, the nonces, and the DH keys. An attacker who can recover a

DH private key can perform an offline attack on the preshared key. Depending on the strength of the PSK, this offline attack may be trivial or may be computationally intractable.[13]

IKEv1, phase 2. After phase 1 completes, there is a second phase, called Quick Mode, which involves another exchange of nonces and, optionally, another DH exchange for forward secrecy [26]. As the messages for phase 2 are protected by the keys established during phase 1, there is no additional encryption. Thus, an attacker who has successfully recovered the phase 1 keys can decrypt phase 2 messages. At this point, if another DH key exchange is used, the attacker can either run the Dual EC-state-recovery attack again or simply walk the Dual EC generator forward to recover the DH private key. If only nonces are exchanged, then no additional work is necessary. In either case, the attacker can compute the traffic keys and recover plain text.

IKEv2, phase 1. A connection in IKEv2 begins by exchanging two request/response pairs which form the initial exchange. The first pair of messages, called IKE_SA_INIT, exchange DH public keys and nonces in the clear. The peers use these to compute a shared secret, SKEYSEED, from which all traffic keys are derived. These keys are used to protect the following messages.

This first exchange contains all of the information necessary for the attacker to recover the Dual EC state and compute a DH private key and thus derive SKEYSEED. This stands in contrast to IKEv1 where the authentication mode influences key derivation and hence, exploitability.

The second exchange, called IKE_AUTH, is encrypted using keys derived from SKEYSEED and is used to authenticate each peer, but plays no role in decryption. At this point, a child security association (CHILD_SA) is set up which can be used for protecting VPN traffic.

IKEv2, phase 2. IKEv2 does contain a second phase, called CREATE_CHILD_SA, which can be used to create additional child security associations. One use of this phase is periodic rekeying. The use of a second phase is optional.[14]

Similar to IKEv1's second phase, nonces and, optionally, DH public keys are exchanged. As before, when DH keys are used, an attacker may either perform the attack to recover Dual EC's state a second time or walk the generator forward.

6. ATTACKING IKE

To validate the attacks we describe above, we purchased a Juniper Secure Services Gateway 550M VPN device, and modified the firmware version 6.3.0r12 in a manner similar to the 2012 attack. This required us to generate a point Q for which we know the trapdoor $(\log_P Q)^{-1}$, and to modify the Dual EC Known Answer Test (KAT) correspondingly. To install the firmware on the device, we further modified a non-cryptographic checksum contained within the header of the firmware.[15]

[13] Anecdotally, the preshared keys used in practice are often quite weak. For example, FlyVPN's "How To Setup L2TP VPN On Android 4" instructs the user to "Input 'vpnserver' letters into 'IPSec pre-shared key.' " https://www.flyvpn.com/How-To-Setup-L2TP-VPN-On-Android-4.html, retrieved February 18, 2016.

[14] "The second request/response (IKE_AUTH) transmits identities, proves knowledge of the secrets corresponding to the two identities, and sets up an SA for the first (and often only) AH and/or ESP CHILD_SA" [26].

[15] If a code-signing certificate is installed on the device, firmware updates require the presence of a valid digital signature on the new firmware using the key in that certificate. Since we did not have a certificate installed, we were able to omit this signature.

Using the new firmware, we next configured the device with three separate VPN gateways: (1) configured for IKEv1 with a PSK, (2) configured for IKEv1 with a 1024-bit RSA signing certificate, and (3) configured for IKEv2 with a PSK. For each configuration, we initiated VPN connections to the box using strongSwan [42]. By capturing the resulting traffic, we were then able to extract the nonces in the IKE handshakes and run the Dual EC attack to recover the state of the random number generator for each connection. As previously discussed, since the 32-byte nonces consist of the concatenation of two consecutive 30-byte Dual EC blocks, truncated to 32-bytes, we used the first 30 bytes of the nonce to recover a potential state value, and then confirmed this guess against the remaining 2 bytes of the nonce.

From this point, we generated a series of Dual EC outputs to obtain a private exponent consistent with the Diffie–Hellman public key observed in the traffic. This required a single modular exponentiation per potential exponent x, followed by a comparison to the extracted key exchange payload value. Given the correct private exponent, we then obtained the shared secret from the initiator value, thereby determining the DH shared secret g^{xy}. Given the Diffie–Hellman shared secret, we implemented the remaining elements of the IKEv1 and IKEv2 standards [21, 26] in order to calculate the Phase 1 (Aggressive Mode) keying material (for IKEv1) and the corresponding IKE_SA_INIT/IKE_AUTH keying material (for IKEv2). This information encrypts the subsequent handshake messages, and is itself used to calculate the key material for subsequent payloads, including Encapsulated Secure Payload (ESP) messages. A challenge in the IKEv1 PSK implementation is the need to incorporate an unknown PSK value into the PRF used to calculate the resulting key material. For our proof of concept implementation we used a known PSK, however without knowledge of this value, an additional brute-force or dictionary attack step would have been required. No such problem exists for the IKEv1 certificate connections, or for IKEv2 PSK.

Using the recovered key material, we next decrypt the remaining traffic, which in each case embeds a second Diffie–Hellman handshake with additional nonces and Diffie–Hellman ephemeral public keys. Since this handshake is also produced from the same generator, we can simply wind the generator forward (or restart with a nonce drawn from the second phase handshake) to recover the corresponding Diffie–Hellman private keys. This new shared secret can then be used to calculate the resulting key material. All subsequent traffic that we see and decrypt utilized the Encapsulating Security Payload (ESP) protocol [29] in tunnel mode.

7. PASSIVELY DETECTING SCREENOS

An adversary who knows the Dual EC Q parameter — either Juniper's 2008 point or the unauthorized 2012 point — may wish to detect vulnerable versions of ScreenOS by passively watching network traffic. In theory, such an adversary has several avenues open to it. The easiest approach is to attempt the attack on every VPN connection to see if the attack is successful. Alternatively, the adversary could attempt to fingerprint VPN boxes and only perform the attack on connections that match.

Dual EC is known to have a small, but nonnegligible, bias. In particular, Schoenmakers and Sidorenko [39] and Gjøsteen [18] give a procedure to distinguish 30-byte blocks generated uniformly at random from those generated by Dual EC. The basic idea is to count how many points on the curve have x-coordinates that agree with the 30-byte block in their least significant 30 bytes. In both the uniformly at random case and the Dual EC case, the number of points on the curve that match follow a normal distribution. In order to use the this distinguisher, one needs to see a sufficient number

of 30-byte blocks (in the form of IKE nonces) to state with high confidence that the blocks came from one distribution or the other.

We empirically computed the parameters of these two distributions to see how difficult this task is. We generated 2 million 30-byte blocks using Dual EC and an additional 2 million blocks uniformly at random and performed the point counting. We estimate the distributions' parameters by fitting the data using maximum likelihood estimation. The results are not encouraging. When generated uniformly at random, the number of points on the curve that agree with the generated block have parameters $\mu = 65536.02$ and $\sigma = 256.05$. When generated using Dual EC, the parameters are $\mu = 65536.78$ and $\sigma = 256.06$. This approach is unlikely to work without seeing tens or hundreds of thousands of connections.

Although detecting the Dual EC bias requires a massive number of connections, the adversary's task is actually much easier due to a bug in ScreenOS's Dual EC implementation. ScreenOS contains a customized version of OpenSSL and uses OpenSSL's elliptic curve and arbitrary-precision (BIGNUM) routines to implement Dual EC. The OpenSSL function to convert a BIGNUM to an array of bytes is `BN_bn2bin()`. Due to a design defect in OpenSSL's API, there is no way to correctly use `BN_bn2bin` without first determining how many bytes it will use — using `BN_num_bytes()` — and zero-padding, and only then using `BN_bn2bin()`:[16]

```
int size = BN_num_bytes(x);
memset(buffer, 0, 30 - size);
BN_bn2bin(x, buffer + (30 - size));
```

ScreenOS's Dual EC implementation omits the zero padding when converting from BIGNUMs to binary output. The upshot is that neither the first nor the thirty-first byte of a nonce will ever be zero.

Thus, if the adversary ever sees a zero byte in either position, it can conclude that the implementation is not Juniper's Dual EC.

If the nonces are generated uniformly at random, then we expect each of these bytes to be zero with (independent) probability 1/256. Thus, the probability that after n nonces without zeros in those positions, the nonce was generated uniformly at random is $(255/256)^{2n}$.

This bug does not affect the exploitability of the Dual EC generator; however, it can lead to a few additional potential internal states to check as described in Section 2.

8. COUNTING SCREENOS DEVICES

The zero-detection technique described in the previous section also suggests a method of counting the number on the Internet running vulnerable versions ScreenOS, as discussed in this section. As a convenience, in the discussion below we say a host *sent a zero* to mean that during an IKEv1 or IKEv2 key exchange, the responder nonce contained a zero at position 0 or 30.

Data collection. We used Internet-wide scans to attempt to measure the population of affected devices. Because the signal to be measured is fairly small, we performed the scan in multiple passes, starting by identifying all hosts which might potentially be running an affected version of ScreenOS and then repeatedly scanning them, dropping each one as soon as it sent a zero, leaving us with a final group of hosts which are highly probable to be ScreenOS. The data collection was performed during April 2016 using ZGrab, an application-layer scanner that operates with ZMap [15].

In the first phase, we performed an Internet-wide scan of all IPv4 addresses on port 500 to determine which hosts were configured to negotiate IKEv1 and IKEv2. Next, we used a custom ZGrab module, which acted as an IKE initiator and attempted to perform the first key exchange phase for both IKEv1 and IKEv2. Our initiator client offered a set of cipher suites chosen from the most commonly supported options observed by Adrian et al. [3].[17] We tested this packet on the default configuration of our vulnerable NetScreen device to verify that it was accepted.

Our initial scan found 7,703,858 hosts that responded to our probes with valid IKE packets, indicating that they support IKEv1 or IKEv2. Of those hosts, only 2,263,314 were willing to perform an initial key exchange with us. The remaining were likely configured to only negotiate with specific whitelisted IPs. Because the affected versions of ScreenOS send 32-byte nonces, we excluded all hosts responding with nonces of other lengths. This left us with 343,467 hosts: 94,201 which just accepted IKEv1, 131,080 which just accepted IKEv2, and 118,186 which accepted both. If a host responded to both IKEv1 and IKEv2, it was subsequently scanned only for IKEv2 in an effort to minimize our impact; see the limitations section below for a discussion of the consequences of this choice.

In the second phase, we periodically scanned all of the remaining hosts that had responded in the preceding scan with a 32-byte nonce without sending a zero. This phase consisted of 601 rounds of scanning, spread out over five days to minimize the impact of our scans on the hosts. By the end of the second phase, 58,695 IKEv1 hosts and 71,360 IKEv2 hosts had stopped responding and were not rescanned. Of the total remaining $N = 213{,}412$ hosts, 31,801 IKEv1 and 176,619 IKEv2 hosts sent a zero, leaving 3,705 IKEv1 hosts and 1,287 IKEv2 hosts that responded to 602 scans (the first phase, baseline scan and the 601 second phase scans) without sending a zero.

If we assume that non-ScreenOS hosts construct nonces uniformly at random — this does not appear to be actually true, as discussed below — the probability that a non-ScreenOS host sent n nonces without sending a zero is $p_n = (255/256)^{2n}$. In particular, the probability that a non-ScreenOS host sent 602 nonces without sending a zero is $p_{602} \approx .90\%$. If S of the N total hosts run ScreenOS, we would expect to see $S + (N - S) \cdot p_{602}$ hosts that do not send a zero after 602 rounds. We measured $3705 + 1287 = 4992$ hosts that did not send a zero and so we estimate that there are about $S \approx 3103$ ScreenOS hosts in our dataset. Using data from Censys [16] scans of other ports running HTTPS, SSH, and/or Telnet management services, we were able to confirm that 447 of these hosts were in fact running some version of ScreenOS.

Limitations. There are several limitations to the data and analyses discussed above: hosts are a weak proxy for traffic, nonpublic hosts, IP blocking, scanning methodology, and analysis methodology. We discuss each in turn.

Fundamentally, we would like to understand how much VPN traffic is vulnerable to Juniper's "knowledgeable attacker." Unfortunately, a network scan cannot answer that question. Instead, we can only compute a weak proxy for this number by determining how many vulnerable VPN devices are on the Internet.

A second potentially-confounding factor is that not all VPN devices are publicly accessible. These devices may be configured to create VPN connections between several private networks and thus would refuse our baseline scan handshake if peer addresses had been explicitly configured in the host. This stands in contrast to a scan to detect vulnerabilities in public web servers using TLS

[16] This defect was corrected quite recently, years after the version of OpenSSL ScreenOS uses was written. https://mta.openssl.org/pipermail/openssl-commits/2016-February/003520.html

[17] We used the 3DES-CBC-SHA, AES-CBC-SHA, AES-CBC-SHA, 3DES-CBC-MD5, DES-CBC-MD5, DES-CBC-SHA, AES-CBC-MD5, and AES-CBC-MD5 cipher suites.

Listing 2: The core ScreenOS 6.1 PRNG subroutine.

```
void prng_generate(char *output) {
  unsigned int index = 0;
  int time[2];

  // FIPS checks removed for clarity
  if (blocks_generated_since_reseed++ > 9999)
    prng_reseed();
  // FIPS checks removed for clarity
  time[0] = 0;
  time[1] = get_cycles();
  do {
    // FIPS checks removed for clarity
    prng_generate_block(time, prng_seed, prng_key,
                        prng_block);
    // FIPS checks removed for clarity
    memcpy(&output[index], prng_output_block,
           min(20-index, 8));
    index += min(20-index, 8);
  } while (index <= 19);
}
```

and represents a fundamental limitation of network mass scans for understanding VPN infrastructure.

Third, our measurements saw a significant drop in response rates over time. Some of this may be due to natural movement of hosts over time, as we attempted to minimize impact by rescanning hosts that had previously responded successfully. In some cases, despite our attempts to scan slowly and over a long period of time, we may have inadvertently triggered ScreenOS's built-in anti-DoS protection for some hosts. We are unable to distinguish these cases.

Fourth, our scanning methodology turned out to be based on flawed assumptions, namely that non-ScreenOS hosts would send uniformly random nonces and that IKEv1 and IKEv2 behavior would be the same. A frequency analysis of the bytes in various positions in the nonces we collected suggests that the bytes are not uniformly distributed in non-ScreenOS hosts (i.e., even nonces from hosts which send zeros are not uniform) and that, for some reason, the nonuniformity is stronger in IKEv2 hosts, making those results even less reliable. Thus, our decision to scan hosts that support both IKEv1 and IKEv2 only using IKEv2 was an error.

9. DISCUSSION

Much attention has been paid to the 2012 compromise of Juniper's ScreenOS source code by unknown parties. In this paper we have shown that the vulnerabilities announced by Juniper can be traced largely to the pre-existing design of Juniper's ScreenOS random number generator. Specifically, we argue that Juniper's design is exploitable due to a series of deliberate design decisions, accidents, and oversights on the part of the ScreenOS developers.

Below we review each of the conditions that are required to produce an exploitable PRNG in the Juniper system:

1. **Implementation of Dual EC.** The cascade design of Juniper's double PRNG, which employs Dual EC to seed the PRNG on each call seems a surprising choice, given the performance limitations of Dual EC. Notably, the transition from ScreenOS 6.1 (X9.31 only) to 6.2 (Dual EC and X9.31) involved the addition of a *nonce pre-generation* queue to the existing DH key queues.[18] One potential motivation for this change could be the additional security assurance provided by Dual EC. However, we note that Juniper did not seek FIPS certification of the Dual EC generator, despite the fact that following the deprecation of the ANSI X9.31 generator on January 1, 2016, it would have been the only FIPS-certified PRNG in their product.[19]
2. **Presence of a Dual EC/ANSI cascade flaw.** Even with Dual EC present in the ScreenOS devices, the use of a cascade between Dual EC and ScreenOS should have prevented the known state recovery attacks. As detailed in Section 4, this protection is not available due to flaws in the cascade implementation, which allows for the exfiltration of unprocessed Dual EC output. This flaw is particularly perplexing. Compare the `prng_generate()` function in version 6.1 (Listing 2) with the analogous function in version 6.2 (Listing 1). Apart from minor changes arising from moving from generating 20 bytes at a time to generating 32 bytes at a time and always reseeding rather than reseeding based on a counter, the functions look quite similar. However, for some reason, the loop index variable was changed from a local variable to a global variable.
3. **Always reseeding.** In version 6.1, ScreenOS reseeded the X9.31 PRNG from system entropy every 10,000 calls (hard-coded; see Listing 2). However, in version 6.2, the reseeding mechanism was repurposed to produce the cascade by always reseeding. When combined with the cascade flaw described above, all PRNG output comes from Dual EC, increasing the probability that a specific value observed by the attacker can be used to recover PRNG state.
4. **Use of 32-byte IKE nonces.** The IKE standards do not provide a specific recommendation for nonce length, stating only that nonces should be between 8 and 256 bytes, and that nonces should be at least half the key size of the PRF used. The last version of ScreenOS without Dual EC was 6.1.0r7 and specified 20 byte nonces. In the subsequent release, ScreenOS 6.2.0r1, Juniper developers added Dual EC and modified the IKE nonce size from 20 to 32 bytes. Efficiently recovering the state of the Dual EC generator requires at a minimum 26 bytes of unprocessed PRNG output, and as discussed in Section 5.2, having greater than 30 bytes expedites the state recovery attack.
5. **Modifying the order of nonce and key generation.** The ScreenOS IKEv1 and IKEv2 implementations both output IKE Key Exchange prior to the IKE Nonce packet. However, this output order does not reflect the *generation* order of the same values in all versions of ScreenOS. In particular, the addition of *nonce queues* in ScreenOS 6.2.0r1 effectively guarantees that in most cases a non-loaded system will generate a nonce immediately prior to the Diffie–Hellman private key that will be used in a given handshake. In practice, this facilitates state recovery attacks that can recover secret keys (and thus enable decryption) within a single IKE handshake, significantly improving the effectiveness of passive attacks.

All told, in the course of a single version revision, Juniper made a series of changes that combined to produce a system which only required the attacker to know the discrete log of Q to be exploitable. See Table 2 for a summary of changes. For a randomly selected Q, or a point chosen using the nothing-up-my-sleeve process proposed in ANSI X9.82 [2], calculating d is likely to be infeasible. We have no way to evaluate the likelihood that some party knows d for Juniper's non-standard Q point, except to note that Juniper does not

[18] To give a rough estimate of the performance difference, we implemented Dual EC and ANSI X9.31 using the same procedure used in ScreenOS and measured how long it takes to generate 32-byte blocks. Dual EC takes roughly 125 times as long as X9.31.

[19] A review of the CMVP certification lists [35] shows that all ScreenOS FIPS certification certificates have indeed been de-listed as of February 2016.

Table 2: ScreenOS features by version.

Version	PRNG	Reseed period (calls)	Reseed bug	DH queue	Nonce queue	Nonce size (bytes)	DH groups supported
6.1.0	X9.31	10000		✓		20	MODP 768, 1024, 1536, 2048
6.2.0	Dual EC + X9.31	1	✓	✓	✓	32	MODP 768, 1024, 1536, 2048
6.3.0	Dual EC + X9.31	1	✓	✓	✓	32	MODP 768, 1024, 1536, 2048; ECP 256, 384

Between versions 6.1.0 and 6.2.0, a cluster of changes were made to the PRNG and IKE subsystems. In the PRNG subsystem, the switch to (1) Dual EC + X9.31; (2) reseeding on every call; and (3) the bug in reseed that causes X9.31 to be skipped produce the necessary conditions to attack IKE. In the IKE subsystem, changing the nonce size from 20 bytes to 32 bytes moves the attack from completely impractical to nearly best-case scenario, from an attacker's point of view. The introduction of a nonce queue changes the nature of the attack such that, in the usual case, an attacker can decrypt a session based solely on that session's traffic.

Version 6.3.0 is nearly identical to 6.2.0 but supports elliptic curve Diffie–Hellman groups. In contrast to the changes between 6.1.0 and 6.2.0, this may actually make an attacker's job harder; see Section 5.5.

appear to have used any of the recommendations presented in the NIST standard [34]. Based on the conclusions of Juniper's 2012 vulnerability report [23], however, it does seem reasonable to assume that the 2012 attacker-generated Q' was maliciously generated.

9.1 Lessons

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

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 256-byte nonces, as permitted by IPsec; it is simply an invitation for implementations to disclose sensitive state, intentionally or not.[20]

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 PSK between the two protocols. It is unfortunate that IKEv2 is easier to exploit.

More generally, protocol designers may wish to hedge their designs against entropy failure [6]. There are few widely deployed cryptographic protocols but many crypto libraries and randomness subsystems, so changes to protocols may do more good than revised design guidelines for PRNGs. One approach is to *derandomize* protocols by replacing randomized nonces with deterministic, cryptographically secure values derived from connection parameters, to avoid nonce collisions or predictability due to improperly seeded random number generators. Bellare et al. [7] propose techniques to render symmetric cryptographic primitives deterministic and resilient against *algorithm substitution attacks*. Future research might investigate whether similar guarantees can be extended to protocols employing asymmetric primitives such as key exchange protocols.

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` routine and the `prng_reseed` routine reuse the same 32-byte buffer, `prng_temporary`, 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.

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.

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.

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 4. NIST should revisit its laboratory accreditation program to ensure more thorough audits, especially of randomness subsystem code.

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 [47], 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-

[20] 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.

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, 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.

For policymakers. In the recent debate over whether law enforcement is "going dark," some have proposed a mandate for cryptographic systems to allow "exceptional access": mechanisms for law enforcement to recover plaintext on demand. Dual EC represents a particularly promising template for such an exceptional access system. Either no one knows the value of d corresponding to NSA's point Q or only NSA does (barring a leak). In NSA's jargon, a Dual EC-style backdoor would be "NOBUS": No one but the U.S. could exploit it.

The unauthorized change to ScreenOS's Dual EC constants made in 2012 illustrates a new threat: the ability for another party to apparently subvert a NOBUS back door for its own purposes, with only minimally detectable changes. Since the Federal Government is a NetScreen customer [36], the changes made in 2008 to add Dual EC to ScreenOS and to expose its output in IKE handshakes may have had negative repercussions for U.S. security, NOBUS or not.

Indeed, despite suspicion, there has been no incontrovertible evidence of Dual EC's being used for kleptography. ScreenOS after the 2012 unauthorized change represents the deployment of Dual EC for which there is the strongest such evidence — and the party that took advantage of it may not have been NSA.

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 2014 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?

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,[21] Juniper has either failed or explicitly refused to provide any further details.

10. RELATED WORK

Dual EC. The history of the Dual EC random number generator was described by Checkoway et al. [9]. By 2006, it was already clear that the generator output has biases in its output that make it unsuitable for deployment, through work by Gjøsteen [18] and by Schoenmakers and Sidorenko [39]. Shumow and Ferguson's presentation at the Crypto 2007 rump session [40] further made clear that someone in possession of the discrete logarithm of Q to base P and who saw raw output from the generator would be able to reconstruct its internal state and predict all future outputs. Nevertheless, Dual EC was adopted as part of NIST's SP 800-90A standard [4], and was not withdrawn until 2015 [5], following reporting based on the Snowden documents [37] that suggested that the Dual EC backdoor might be intentional. A presentation by John Kelsey gives a postmortem of Dual EC standardization from NIST's perspective [27].

Our analysis in Section 4 shows that Juniper adopted Dual EC in 2008. In 2013, NIST's reopening SP 800-90A for comments led Juniper to publish a knowledge base article explaining that ScreenOS uses Dual EC, but "in a way that should not be vulnerable to the possible issue that has been brought to light," because of the custom Q and because Dual EC output is filtered through X9.31 [25]. As our analysis shows, at the time that Juniper made this statement, the Q value shipping in ScreenOS was *already* one introduced in the unauthorized 2012 change. In January 2016, Juniper announced that it would remove Dual EC from its ScreenOS products in "the first half of 2016" [44].

Randomness failures. Many instances of randomness failures in widely deployed systems have been reported. In 1996, Goldberg and Wagner showed that the Netscape browser seeded its PRNG insecurely, allowing SSL traffic to be decrypted [19].

Between 2006 and and 2008, Linux systems running the Debian distribution or its derivatives (including Ubuntu) shipped a modified version of the OpenSSL library that failed to incorporate entropy from the kernel into its own entropy pool. The available entropy was then low enough under normal conditions that the keys that affected systems generate could be exhaustively enumerated and identified over the network [45].

Heninger et al. [22] performed a pairwise GCD on RSA moduli obtained from scanning the IPv4 address space, finding many shared factors and weak keys; the root cause was the lack of entropy available shortly after boot in many network devices. Kim et al. [31] showed that a related problem affected OpenSSL on Android.

Bernstein et al. showed that randomness failures in smart cards allowed private keys to be recovered using lattice attacks [8].

Design of PRNGs. A line of work beginning with Kelsey et al. [28] and continuing to today [11, 12] has sought to formalize the security desiderata for PRNGs used as part of cryptographic systems, and to evaluate deployed PRNGs against these desiderata. Gutterman et al. [20] and, later, Lacharme et al. [32] analyzed the Linux randomness system; Dorrendorf et al. [14] analyzed that of Windows.

As new use cases arose, the security desiderata have been revised and expanded. For example, Ristenpart and Yilek analyzed application-level randomness reuse in virtual machines whose state is reset and rolled back [38], and Everspaugh et al. [17] extended the analysis to kernel-level randomness.

Kleptography. Young and Yung formalized a theoretical model of cryptographic backdoors in black box cryptography that they called "kleptography" [47]. More recently, Bellare, Paterson, and Rogaway developed a model of algorithmic substitution attacks as a formalization of a cryptographic back door and designed symmetric encryption schemes to secure against this forms of attacks [7]. Dodis et al. provide a formal treatment of backdoored PRNGs [13].

Stevens developed techniques for "counter-cryptanalysis" that he used to reconstruct the MD5 collision attack that the unknown authors of the Flame malware exploited against the Microsoft Terminal Server Licensing Service prior to its discovery in 2012 [41]. This incident is the best prior example we have of a publicly visible cryptanalytic attack carried out by sophisticated attackers.

[21] Online: https://oversight.house.gov/hearing/federal-cybersecurity-detection-response-and-mitigation/.

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] Accredited Standards Committee (ASC) X9, Financial Services. ANS X9.31-1998: Digital signatures using reversible algorithms for the financial services industry (rDSA), 1998. Withdrawn.

[2] Accredited Standards Committee (ASC) X9, Financial Services. ANS X9.82-3-2007: Random number generation, part 3: Deterministic random bit generators, 2007.

[3] D. Adrian, K. Bhargavan, Z. Durumeric, P. Gaudry, M. Green, J. A. Halderman, N. Heninger, D. Springall, E. Thomé, L. Valenta, B. VanderSloot, E. Wustrow, S. Zanella-Béguelin, and P. Zimmermann. Imperfect forward secrecy: How Diffie-Hellman fails in practice. In C. Kruegel and N. Li, editors, *Proceedings of CCS 2015*, pages 5–17. ACM Press, Oct. 2015.

[4] E. Barker and J. Kelsey. NIST Special Publication 800-90A: Recommendation for Random Number Generation Using Deterministic Random Bit Generators. Technical report, National Institute of Standards and Technology, 2006.

[5] E. Barker and J. Kelsey. NIST Special Publication 800-90A Revision 1: Recommendation for Random Number Generation Using Deterministic Random Bit Generators. Technical report, National Institute of Standards and Technology, June 2015.

[6] M. Bellare, Z. Brakerski, M. Naor, T. Ristenpart, G. Segev, H. Shacham, and S. Yilek. Hedged public-key encryption: How to protect against bad randomness. In M. Matsui, editor, *Proceedings of Asiacrypt 2009*, volume 5912 of *LNCS*, pages 232–49. Springer-Verlag, Dec. 2009.

[7] M. Bellare, K. G. Paterson, and P. Rogaway. Security of symmetric encryption against mass surveillance. In J. Garay and R. Gennaro, editors, *Proceedings of Crypto 2014, Part I*, volume 8616 of *LNCS*, pages 1–19. Springer-Verlag, Aug. 2014.

[8] D. J. Bernstein, Y.-A. Chang, C.-M. Cheng, L.-P. Chou, N. Heninger, T. Lange, and N. Someren. Factoring RSA Keys from Certified Smart Cards: Coppersmith in the Wild. In K. Sako and P. Sarkar, editors, *Proceedings of Asiacrypt 2013*, volume 8270 of *LNCS*, pages 341–60. Springer-Verlag, Dec. 2013.

[9] S. Checkoway, M. Fredrikson, R. Niederhagen, A. Everspaugh, M. Green, T. Lange, T. Ristenpart, D. J. Bernstein, J. Maskiewicz, and H. Shacham. On the practical exploitability of Dual EC in TLS implementations. In K. Fu, editor, *Proceedings of USENIX Security 2014*, pages 319–35. USENIX, Aug. 2014.

[10] J. R. Clapper. Worldwide threat assessment of the U.S. intelligence community. Statement for the record, Senate Armed Services Committee. Online: http://www.armed-services.senate.gov/imo/media/doc/Clapper_02-09-16.pdf, Feb. 2016.

[11] Y. Dodis, D. Pointcheval, S. Ruhault, D. Vergnaud, and D. Wichs. Security analysis of pseudo-random number generators with input: /dev/random is not robust. In V. Gligor and M. Yung, editors, *Proceedings of CCS 2013*, pages 647–58. ACM Press, Nov. 2013.

[12] Y. Dodis, A. Shamir, N. Stephens-Davidowitz, and D. Wichs. How to eat your entropy and have it too — optimal recovery strategies for compromised RNGs. In J. Garay and R. Gennaro, editors, *Proceedings of Crypto 2014, Part II*, volume 8617 of *LNCS*, pages 37–54. Springer-Verlag, Aug. 2014.

[13] Y. Dodis, C. Ganesh, A. Golovnev, A. Juels, and T. Ristenpart. A formal treatment of backdoored pseudorandom generators. In M. Fischlin and E. Oswald, editors, *Proceedings of EUROCRYPT 2015*, pages 101–126. Springer, Apr. 2015.

[14] L. Dorrendorf, Z. Gutterman, and B. Pinkas. Cryptanalysis of the random number generator of the Windows operating system. *ACM Trans. Info. & System Security*, 13(1):10, 2009.

[15] Z. Durumeric, E. Wustrow, and J. A. Halderman. ZMap: Fast Internet-wide scanning and its security applications. In S. King, editor, *Proceedings of USENIX Security 2013*, pages 605–619. USENIX, Aug. 2013.

[16] Z. Durumeric, D. Adrian, A. Mirian, M. Bailey, and J. A. Halderman. A search engine backed by Internet-wide scanning. In C. Kruegel and N. Li, editors, *Proceedings of CCS 2015*, pages 542–53. ACM Press, Oct. 2015.

[17] A. Everspaugh, Y. Zhai, R. Jellinek, T. Ristenpart, and M. Swift. Not-so-random numbers in virtualized Linux and the Whirlwind RNG. In M. Backes, A. Perrig, and H. Wang, editors, *Proceedings of Security and Privacy ("Oakland") 2014*, pages 559–74. IEEE Computer Society, May 2014.

[18] K. Gjøsteen. Comments on Dual-EC-DRBG/NIST SP 800-90, draft December 2005. Online: https://www.math.ntnu.no/~kristiag/drafts/dual-ec-drbg-comments.pdf, Mar. 2006.

[19] I. Goldberg and D. Wagner. Randomness and the Netscape browser. *Dr. Dobb's Journal*, 21(1):66–70, Jan. 1996.

[20] Z. Gutterman, B. Pinkas, and T. Reinman. Analysis of the Linux random number generator. In V. Paxson and B. Pfitzmann, editors, *Proceedings of Security and Privacy ("Oakland") 2006*, pages 371–85. IEEE Computer Society, May 2006.

[21] D. Harkins and D. Carrel. The Internet Key Exchange (IKE). RFC 2409 (Proposed Standard), Nov. 1998. Obsoleted by RFC 4306, updated by RFC 4109. Online: https://tools.ietf.org/html/rfc2409.

[22] N. Heninger, Z. Durumeric, E. Wustrow, and J. A. Halderman. Mining your Ps and Qs: Detection of widespread weak keys in network devices. In T. Kohno, editor, *Proceedings of USENIX Security 2012*. USENIX, Aug. 2012.

[23] Juniper Networks. 2015-12 Out of Cycle Security Bulletin: ScreenOS: Multiple Security issues with ScreenOS (CVE-2015-7755, CVE-2015-7756), Dec. 15. URL https://kb.juniper.net/InfoCenter/index?page=content&id=JSA10713&cat=SIRT_1&actp=LIST.

[24] Juniper Networks. *Concepts & Examples ScreenOS Reference Guide: Virtual Private Networks*, rev. 02 edition, Dec. 2012. URL http://www.juniper.net/techpubs/software/screenos/screenos6.3.0/630_ce_VPN.pdf.

[25] 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&pmv=print&actp=LIST.

[26] C. Kaufman. Internet Key Exchange (IKEv2) Protocol. RFC 4306 (Proposed Standard), Dec. 2005. Obsoleted by RFC 5996, updated by RFC 5282. Online: https://tools.ietf.org/html/rfc4306.

[27] J. Kelsey. 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.

[28] J. Kelsey, B. Schneier, D. Wagner, and C. Hall. Cryptanalytic attacks on pseudorandom number generators. In S. Vaudenay, editor, *Proceedings of FSE 1998*, volume 1372 of *LNCS*, pages 168–88. Springer-Verlag, Mar. 1998.

[29] S. Kent. IP Encapsulating Security Payload (ESP). RFC 4303 (Proposed Standard), Nov. 2005. Online: https://tools.ietf.org/html/rfc4303.

[30] S. Kent and K. Seo. Security architecture for the Internet Protocol. RFC 4301 (Proposed Standard), Dec. 2005. Online: https://tools.ietf.org/html/rfc4301.

[31] S. H. Kim, D. Han, and D. H. Lee. Predictability of Android OpenSSL's pseudo random number generator. In V. Gligor and M. Yung, editors, *Proceedings of CCS 2013*, pages 659–68. ACM Press, Nov. 2013.

[32] P. Lacharme, A. Röck, V. Strubel, and M. Videau. The Linux pseudorandom number generator revisited. Cryptology ePrint Archive, Report 2012/251, 2012. https://eprint.iacr.org/.

[33] H. D. Moore. CVE-2015-7755: Juniper ScreenOS Authentication Backdoor. https://community.rapid7.com/community/infosec/blog/2015/12/20/cve-2015-7755-juniper-screenos-authentication-backdoor, Dec. 2015.

[34] 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.

[35] National Institute of Standards and Technology. CMVP historical validation list, Feb. 2016. URL http://csrc.nist.gov/groups/STM/cmvp/documents/140-1/140val-historical.htm. Retrieved February 18, 2016.

[36] Office of Personnel Management. Juniper network firewall maintenance renewal. FedBizOps.gov solicitation number M-13-00031. Online: https://www.fbo.gov/index?id=b3246ffee0a3e9c0ced948b3a8ebca7b, Sept. 2013.

[37] N. Perlroth, J. Larson, and S. Shane. 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.

[38] T. Ristenpart and S. Yilek. When good randomness goes bad: Virtual machine reset vulnerabilities and hedging deployed cryptography. In W. Lee, editor, *Proceedings of NDSS 2010*. Internet Society, Feb. 2010.

[39] B. Schoenmakers and A. Sidorenko. Cryptanalysis of the Dual Elliptic Curve pseudorandom generator. Cryptology ePrint Archive, Report 2006/190, 2006. URL https://eprint.iacr.org/.

[40] D. Shumow and N. Ferguson. 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.yp.to/15-shumow.pdf.

[41] M. Stevens. Counter-cryptanalysis. In C. Ran and J. A. Garay, editors, *Proceedings of Crypto 2013, Part I*, volume 8042 of *LNCS*, pages 129–46. Springer-Verlag, Aug. 2013.

[42] strongSwan. strongSwan: the opensource IPsec-based VPN solution, Nov. 2015. URL https://www.strongswan.org/.

[43] R.-P. Weinmann. Some analysis of the backdoored backdoor. Online: https://rpw.sh/blog/2015/12/21/the-backdoored-backdoor/, Dec. 2015.

[44] B. Worrall. Advancing the security of Juniper products. Online: http://forums.juniper.net/t5/Security-Incident-Response/Advancing-the-Security-of-Juniper-Products/ba-p/286383, Jan. 2016.

[45] S. Yilek, E. Rescorla, H. Shacham, B. Enright, and S. Savage. When private keys are public: Results from the 2008 Debian OpenSSL vulnerability. In A. Feldmann and L. Mathy, editors, *Proceedings of IMC 2009*, pages 15–27. ACM Press, Nov. 2009.

[46] T. Ylonen and C. Lonvick. The Secure Shell (SSH) Protocol Architecture. RFC 4251 (Proposed Standard), Jan. 2006. Online: https://tools.ietf.org/html/rfc4251.

[47] A. Young and M. Yung. Kleptography: Using cryptography against cryptography. In W. Fumy, editor, *Proceedings of Eurocrypt 1997*, volume 1233 of *LNCS*, pages 62–74. Springer-Verlag, May 1997.