Oracle Corporation

Solaris 11.4

Assurance Activity Report

Version 1.3

February 8, 2021

Document prepared by

www.lightshipsec.com

Table of Contents

1 Introduction

1 This Assurance Activity Report (AAR) documents the evaluation activities performed by Lightship Security for the evaluation identified in Table 1. The AAR is produced in accordance with National Information Assurance Program (NIAP) reporting guidelines.

1.1 Evaluation Identifiers

Table 1: Evaluation Identifiers

Scheme	Canadian Common Criteria Scheme
Evaluation Facility	Lightship Security
Developer/Sponsor	Oracle Corporation
TOE	Solaris 11.4 Build: 11.4 SRU 26.0.1 with IDR 4534 v3
Security Target	[ST] Oracle Solaris 11.4 Security Target, v1.3, February 2021
Protection Profile	[OSPP] Protection Profile for General Purpose Operating Systems, v4.2.1 [SSHEP] Extended Package for Secure Shell (SSH), v1.0

1.2 Evaluation Methods

2 The evaluation was performed using the methods, tools and standards identified in Table 2.

Table 2: Evaluation Methods

Evaluation Criteria	CC v3.1R5
Evaluation Methodology	CEM v3.1R5
Interpretations	

TD #	Name
0525	Updates to Certificate Revocation (FIA_X509_EXT.1)
0501	Cryptographic selections and updates for OS PP
0496	GPOS PP adds allow-with statement for VPN Client V2.1
0493	X.509v3 certificates when using digital signatures for Boot Integrity
0463	Clarification for FPT_TUD_EXT

	0446	Missing selections for SSH
	0441	Updated TLS Ciphersuites for OS PP
	0420	Conflict in FCS_SSHC_EXT.1.1 and FCS_SSHS_EXT.1.1
	0386	Platform-Provided Verification of Update
	0365	FCS_CKM_EXT.4 selections
	0332	Support for RSA SHA2 host keys
	0331	SSH Rekey Testing
	0240	FCS_COP.1.1(1) Platform provided crypto for encryption/decryption
Tools	Please refer to Test Report	

2 TOE Details

2.1 Overview

1 Oracle Solaris is a UNIX-based operating system designed to deliver a consistent platform to run enterprise applications.

2.2 TOE Models and Platforms

2.2.1 Software

2 The TOE encompasses the following software:

a) Solaris 11.4 Build: 11.4 SRU 26.0.1 (11.4-11.4.26.0.1.75.4) with IDR 4534 v3

2.2.2 Hardware

3 The evaluated configuration includes the hardware platforms shown in Table 3.

Table 3: Hardware Platforms

Model	CPU
Oracle SPARC T8-2	SPARC M8
Oracle Server X8-2	Intel Xeon Gold 5200 series

2.3 Reference Documents

Table 4: List of Reference Documents

Ref	Document
[ST]	Oracle Solaris 11.4 Security Target, v1.3, February 2021
[SUPP]	Oracle Solaris 11.4 Common Criteria Guide, v1.3, January 2021
[INFO]	Oracle Solaris 11.4 Information Library - https://docs.oracle.com/cd/E37838_01/

2.4 Summary of SFRs

Table 5: List of SFRs

Requirement	Title
FAU_GEN.1	Audit Data Generation (Refined)
FCS_CKM.1	Cryptographic Key Generation (Refined)
FCS_CKM.2	Cryptographic Key Establishment (Refined)
FCS_CKM_EXT.4	Cryptographic Key Destruction

Requirement	Title
FCS_COP.1(1)	Cryptographic Operation - Encryption/Decryption (Refined)
FCS_COP.1(2)	Cryptographic Operation - Hashing (Refined)
FCS_COP.1(3)	Cryptographic Operation - Signing (Refined)
FCS_COP.1(4)	Cryptographic Operation - Keyed-Hash Message Authentication (Refined)
FCS_COP.1/SSH	FCS_COP.1/SSH Cryptographic Operation - Encryption/Decryption (Refined)
FCS_RBG_EXT.1	Random Bit Generation
FCS_SSH_EXT.1	SSH Protocol
FCS_SSHS_EXT.1	SSH Protocol - Server
FCS_STO_EXT.1	Storage of Sensitive Data
FCS_TLSC_EXT.1	TLS Client Protocol
FDP_ACF_EXT.1	Access Controls for Protecting User Data
FIA_AFL.1	Authentication Failure Handling (Refined)
FIA_UAU.5	Multiple Authentication Mechanisms (Refined)
FIA_X509_EXT.1	X.509 Certificate Validation
FIA_X509_EXT.2	X.509 Certificate Authentication
FMT_MOF_EXT.1	Management of security functions behavior
FMT_SMF_EXT.1	Specification of Management Functions
FPT_ACF_EXT.1	Access controls
FPT_ASLR_EXT.1	Address Space Layout Randomization
FPT_SBOP_EXT.1	Stack Buffer Overflow Protection
FPT_TST_EXT.1	Boot Integrity
FPT_TUD_EXT.1	Trusted Update
FPT_TUD_EXT.2	Trusted Update for Application Software
FTP_ITC_EXT.1	Trusted channel communication
FTP_TRP.1	Trusted Path

3 Evaluation Activities for SFRs

3.1 Cryptographic Support (FCS)

3.1.1 FCS_CKM.1 Cryptographic Key Generation (Refined)

3.1.1.1 TSS

3 The evaluator will ensure that the TSS identifies the key sizes supported by the OS. If the ST specifies more than one scheme, the evaluator will examine the TSS to verify that it identifies the usage for each scheme.

Findings:	Section 6.2.1 of the [ST] specifies that the TOE supports the following schemes and key sizes: RSA (2048, 3072), ECDSA (P-256, P-384, P-521), FFC safe primes.

3.1.1.2 Guidance Documentation

4 The evaluator will verify that the AGD guidance instructs the administrator how to configure the OS to use the selected key generation scheme(s) and key size(s) for all uses defined in this PP.

Findings:	In section 3.4.1 of the [SUPP], the guidance instructs the administrator to only use appropriate RSA and ECDSA key generation mechanisms for the purpose of constructing SSH hostkey private keys.

3.1.1.3 Tests

5 Evaluation Activity Note: The following tests may require the vendor to furnish a developer environment and developer tools that are typically not available to end-users of the OS.

6 **Key Generation for FIPS PUB 186-4 RSA Schemes**

7 The evaluator will verify the implementation of RSA Key Generation by the OS using the Key Generation test. This test verifies the ability of the TSF to correctly produce values for the key components including the public verification exponent e, the private prime factors p and q, the public modulus n and the calculation of the private signature exponent d. Key Pair generation specifies 5 ways (or methods) to generate the primes p and q. These include:

8 1. Random Primes:

- Provable primes
- Probable primes

9 2. Primes with Conditions:

- Primes p1, p2, q1,q2, p and q shall all be provable primes
- Primes p1, p2, q1, and q2 shall be provable primes and p and q shall be probable primes
- Primes p1, p2, q1,q2, p and q shall all be probable primes

10 To test the key generation method for the Random Provable primes method and for all the Primes with Conditions methods, the evaluator must seed the TSF key generation routine with sufficient data to deterministically generate the RSA key pair. This includes the random seed(s), the public exponent of the RSA key, and the desired key length. For each key length supported, the evaluator shall have the TSF generate 25 key pairs. The evaluator will verify the correctness of the TSF's implementation by comparing values generated by the TSF with those generated from a known good implementation.

11 If possible, the Random Probable primes method should also be verified against a known good implementation as described above. Otherwise, the evaluator will have the TSF generate 10 keys pairs for each supported key length nlen and verify:

- $n = p \cdot q$,
- p and q are probably prime according to Miller-Rabin tests,
- $GCD(p-1,e) = 1$,
- $GCD(q-1,e) = 1$,
- $216 \leq e \leq 2256$ and e is an odd integer,
- $|p-q| > 2nlen/2 - 100$,
- $p \geq 2nlen/2 -1/2$,
- $q \geq 2nlen/2 -1/2$,
- $2(nlen/2) < d < LCM(p-1,q-1)$,
- $e \cdot d = 1 \bmod LCM(p-1,q-1)$.

Findings:	The evaluator found that RSA KeyGen (186-4) was tested on the available platforms for CAVP C1651 https://csrc.nist.gov/projects/cryptographic-algorithm-validation-program/details?product=12239.

12 **Key Generation for Elliptic Curve Cryptography (ECC)**

13 FIPS 186-4 ECC Key Generation Test

14 For each supported NIST curve, i.e., P-256, P-384 and P-521, the evaluator will require the implementation under test (IUT) to generate 10 private/public key pairs. The private key shall be generated using an approved random bit generator (RBG). To determine correctness, the evaluator will submit the generated key pairs to the public key verification (PKV) function of a known good implementation.

15 FIPS 186-4 Public Key Verification (PKV) Test

16 For each supported NIST curve, i.e., P-256, P-384 and P-521, the evaluator will generate 10 private/public key pairs using the key generation function of a known good implementation and modify five of the public key values so that they are incorrect, leaving five values unchanged (i.e., correct). The evaluator will obtain in response a set of 10 PASS/FAIL values.

Findings:	The evaluator found that ECDSA KeyGen (186-4) and KeyVer (186-4) were tested on the available platforms for CAVP C1651 https://csrc.nist.gov/projects/cryptographic-algorithm-validation-program/details?product=12239.

17 **Key Generation for Finite-Field Cryptography (FFC)**

18 The evaluator will verify the implementation of the Parameters Generation and the Key Generation for FFC by the TOE using the Parameter Generation and Key Generation test. This test verifies the ability of the TSF to correctly produce values for the field prime p, the cryptographic prime q (dividing p-1), the cryptographic group generator g, and the calculation of the private key x and public key y.

19 The Parameter generation specifies 2 ways (or methods) to generate the cryptographic prime q and the field prime p:

20 • Cryptographic and Field Primes:

- Primes q and p shall both be provable primes
- Primes q and field prime p shall both be probable primes

21 and two ways to generate the cryptographic group generator g:

22 • Cryptographic Group Generator:

- Generator g constructed through a verifiable process
- Generator g constructed through an unverifiable process

23 The Key generation specifies 2 ways to generate the private key x:

24 • Private Key:

- len(q) bit output of RBG where $1 \leq x \leq q-1$
- len(q) + 64 bit output of RBG, followed by a mod q-1 operation where $1 \leq x \leq q-1$

25 The security strength of the RBG must be at least that of the security offered by the FFC parameter set. To test the cryptographic and field prime generation method for the provable primes method and/or the group generator g for a verifiable process, the evaluator must seed the TSF parameter generation routine with sufficient data to deterministically generate the parameter set. For each key length supported, the evaluator will have the TSF generate 25 parameter sets and key pairs. The evaluator will verify the correctness of the TSF's implementation by comparing values generated by the TSF with those generated from a known good implementation. Verification must also confirm:

- g != 0,1
- q divides p-1
- $g^q \bmod p = 1$
- $g^x \bmod p = y$
- for each FFC parameter set and key pair.

Findings:	No testing required as the TOE does not claim general-purpose FFC. It only claims FFC with safe primes, which does not require CAVP testing.

26 **NIAP TD0501**

27 **Diffie-Hellman Group 14 and FFC Schemes using "safe-prime" groups**

28 Testing for FFC Schemes using Diffie-Hellman group 14 and/or "safe-prime" groups is done as part of testing in FCS_CKM.2.1

Findings:	Please refer to FCS_CKM.2 for descriptions of testing DH group 14 for safe prime groups.

3.1.2 FCS_CKM.2 Cryptographic Key Establishment (Refined)

3.1.2.1 TSS

29 The evaluator will ensure that the supported key establishment schemes correspond to the key generation schemes identified in FCS_CKM.1.1. If the ST specifies more than one scheme, the evaluator will examine the TSS to verify that it identifies the usage for each scheme.

Findings:	Section 6.2.1 of the [ST] specifies that the TOE uses RSA and FFC-based (safe primes only) schemes in TLS and SSH. Section 6.2.1 of the [ST] provides a table which breaks down the scheme per use in the claimed protocols.

30 The evaluator will verify that the TSS describes whether the OS acts as a sender, a recipient, or both for RSA-based key establishment schemes.

Findings:	Section 6.2.1 of the [ST] specifies the TOE acts as the sender for RSA based key establishment schemes.

31 The evaluator will ensure that the TSS describes how the OS handles decryption errors. In accordance with NIST Special Publication 800-56B, the OS must not reveal the particular error that occurred, either through the contents of any outputted or logged error message or through timing variations.

Findings:	Section 6.2.1 of the [ST] specifies that in the event of a decryption error, the OS only logs/outputs aggregate generic error messages.

3.1.2.2 Guidance Documentation

32 The evaluator will verify that the AGD guidance instructs the administrator how to configure the OS to use the selected key establishment scheme(s).

Findings:	In section 3.4.1 of the [SUPP], the administrator is instructed to make use of appropriate key exchange algorithms for the SSH daemon. In section 3.4.2 of [SUPP], appropriate developers are directed to review the man pages for OpenSSL in order to configure the appropriate key establishment ciphers. The set of approved ciphersuites are given in 3.4.2 of [SUPP].

3.1.2.3 Tests

33 Evaluation Activity Note: The following tests require the developer to provide access to a test platform that provides the evaluator with tools that are typically not found on factory products.

34 **Key Establishment Schemes**

35 The evaluator will verify the implementation of the key establishment schemes supported by the OS using the applicable tests below.

36 **SP800-56A Key Establishment Schemes**

37 The evaluator will verify the OS's implementation of SP800-56A key agreement schemes using the following Function and Validity tests. These validation tests for each key agreement scheme verify that the OS has implemented the components of the key agreement scheme according to the specifications in the Recommendation. These components include the calculation of the discrete logarithm cryptography (DLC) primitives (the shared secret value Z) and the calculation of the derived keying material (DKM) via the Key Derivation Function (KDF). If key confirmation is supported, the evaluator will also verify that the components of key confirmation have been implemented correctly, using the test procedures described below. This includes the parsing of the DKM, the generation of MAC data and the calculation of MAC tag.

38 **Function Test**

39 The Function test verifies the ability of the OS to implement the key agreement schemes correctly. To conduct this test the evaluator will generate or obtain test vectors from a known good implementation of the OS's supported schemes. For each supported key agreement scheme-key agreement role combination, KDF type, and, if supported, key confirmation role- key confirmation type combination, the tester shall generate 10 sets of test vectors. The data set consists of the NIST approved curve (ECC) per 10 sets of public keys. These keys are static, ephemeral or both depending on the scheme being tested.

40 The evaluator will obtain the DKM, the corresponding OS's public keys (static and/or ephemeral), the MAC tag(s), and any inputs used in the KDF, such as the Other Information field OI and OS id fields.

41 If the OS does not use a KDF defined in SP 800-56A, the evaluator will obtain only the public keys and the hashed value of the shared secret.

42 The evaluator will verify the correctness of the TSF's implementation of a given scheme by using a known good implementation to calculate the shared secret value, derive the keying material DKM, and compare hashes or MAC tags generated from these values.

43 If key confirmation is supported, the OS shall perform the above for each implemented approved MAC algorithm.

44 **Validity Test**

45 The Validity test verifies the ability of the OS to recognize another party's valid and invalid key agreement results with or without key confirmation. To conduct this test, the evaluator will obtain a list of the supporting cryptographic functions included in the SP800-56A key agreement implementation to determine which errors the OS should be able to recognize. The evaluator generates a set of 30 test vectors consisting of data sets including domain parameter values or NIST approved curves, the evaluator's public keys, the OS's public/private key pairs, MAC tag, and any inputs used in the KDF, such as the other info and OS id fields.

46 The evaluator will inject an error in some of the test vectors to test that the OS recognizes invalid key agreement results caused by the following fields being incorrect: the shared secret value Z, the DKM, the other information field OI, the data to be MAC'd, or the generated MAC tag. If the OS contains the full or partial (only ECC) public key validation, the evaluator will also individually inject errors in both parties' static public keys, both parties' ephemeral public keys and the OS's static

private key to assure the OS detects errors in the public key validation function and/or the partial key validation function (in ECC only). At least two of the test vectors shall remain unmodified and therefore should result in valid key agreement results (they should pass).

47 The OS shall use these modified test vectors to emulate the key agreement scheme using the corresponding parameters. The evaluator will compare the OS's results with the results using a known good implementation verifying that the OS detects these errors.

Findings:	For KAS-FFC-based safe prime groups, testing is per TD 501 (below).

48 **TD0501 - ~~SP800-56B Key Establishment Schemes~~**

49 **TD0501 - RSAES-PKCS1-v1_5 Key Establishment Schemes**

50 The evaluator shall verify the correctness of the TSF's implementation of RSAES-PKCS1-v1_5 by using a known good implementation for each protocol selected in FTP_ITC_EXT.1 that uses RSAES-PKCS1-v1_5.

Findings:	Please see FCS_TLSC_EXT.1 for confirmation.

51 **Diffie-Hellman Group 14**

52 The evaluator shall verify the correctness of the TSF's implementation of Diffie-Hellman group 14 by using a known good implementation for each protocol selected in FTP_ITC_EXT.1 that uses Diffie-Hellman Group 14.

Findings:	Please see FCS_SSHS_EXT.1 for confirmation.

53 **TD0501 - FFC Schemes using "safe-prime" groups (identified in Appendix D of SP 800-56A Revision 3)**

54 The evaluator shall verify the correctness of the TSF's implementation of "safe-prime" groups by using a known good implementation for each protocol selected in FTP_ITC_EXT.1 that uses "safe-prime" groups. This test must be performed for each "safe-prime" group that each protocol uses.

Findings:	Please see FCS_TLSC_EXT.1 test case 1 for confirmation of DHE ciphersuites using safe-prime groups.

3.1.3 FCS_CKM_EXT.4 Cryptographic Key Destruction

3.1.3.1 TSS

55 **Entire section modified by TD0365**

56 The evaluator examines the TSS to ensure it describes how the keys are managed in volatile memory. This description includes details of how each identified key is

introduced into volatile memory (e.g. by derivation from user input, or by unwrapping a wrapped key stored in non-volatile memory) and how they are overwritten.

Findings:	In section 6.2.2 of the [ST], the table shows each key, its generator (i.e. how the key is created), where the key material is stored and its method of destruction. The generator/initiator column represents how the key is introduced into volatile memory. Of specific note, the ZFS key encryption keys can be derived or they can be raw keys. The ZFS data encryption keys are wrapped by key encryption keys. For all keys in volatile memory, the claimed method of zeroization is to remove power from the device.

57 The evaluator will check to ensure the TSS lists each type of key that is stored in in non-volatile memory, and identifies how the TOE interacts with the underlying platform to manage keys (e.g., store, retrieve, destroy). The description includes details on the method of how the TOE interacts with the platform, including an identification and description of the interfaces it uses to manage keys (e.g., file system APIs, platform key store APIs).

Findings:	In section 6.2.2 of the [ST], the table shows each key, its generator (i.e. how the key is created), where the key material is stored and its method of destruction. The generator/initiator column represents how the key is introduced into non-volatile memory (i.e. persistent keys).

58 If the ST makes use of the open assignment and fills in the type of pattern that is used, the evaluator examines the TSS to ensure it describes how that pattern is obtained and used. The evaluator will verify that the pattern does not contain any CSPs.

Findings:	The TOE does not claim an overwrite pattern.

59 The evaluator will check that the TSS identifies any configurations or circumstances that may not strictly conform to the key destruction requirement.

Findings:	The TSS does not identify any configurations or circumstances that do not strictly conform with the key destruction requirements.

60 If the selection “destruction of all key encrypting keys protecting target key according to FCS_CKM_EXT.4.1, where none of the KEKs protecting the target key are derived” is included the evaluator shall examine the TOE’s keychain in the TSS and identify each instance when a key is destroyed by this method. In each instance the evaluator shall verify all keys capable of decrypting the target key are destroyed in accordance with a specified key destruction method in FCS_CKM_EXT.4.1 The evaluator shall verify that all of the keys capable of decrypting the target key are not able to be derived to re-establish the keychain after their destruction.

Findings:	The TOE does not make this claim.

3.1.3.2 Guidance Documentation

61 **Entire section modified by TD0365**

62 There are a variety of concerns that may prevent or delay key destruction in some cases. The evaluator will check that the guidance documentation identifies configurations or circumstances that may not strictly conform to the key destruction requirement, and that this description is consistent with the relevant parts of the TSS

and any other relevant Required Supplementary Information. The evaluator will check that the guidance documentation provides guidance on situations where key destruction may be delayed at the physical layer and how such situations can be avoided or mitigated if possible.

63 Some examples of what is expected to be in the documentation are provided here.

64 When the TOE does not have full access to the physical memory, it is possible that the storage may be implementing wear-leveling and garbage collection. This may create additional copies of the key that are logically inaccessible but persist physically. In this case, to mitigate this the drive should support the TRIM command and implements garbage collection to destroy these persistent copies when not actively engaged in other tasks.

65 Drive vendors implement garbage collection in a variety of different ways, as such there is a variable amount of time until data is truly removed from these solutions. There is a risk that data may persist for a longer amount of time if it is contained in a block with other data not ready for erasure. To reduce this risk, the operating system and file system of the OE should support TRIM, instructing the non-volatile memory to erase copies via garbage collection upon their deletion. If a RAID array is being used, only set-ups that support TRIM are utilized. If the drive is connected via PCI-Express, the operating system supports TRIM over that channel.

66 The drive should be healthy and contains minimal corrupted data and should be end-of-lifed before a significant amount of damage to drive health occurs, this minimizes the risk that small amounts of potentially recoverable data may remain in damaged areas of the drive.

Findings:	The guidance documentation does not readily identify any such circumstances where key destruction is delayed at the physical layer. The TOE is careful to claim that their responsibility lay in deleting the abstraction that represents the persistent key in FCS_CKM_EXT.4. This pushes ultimate responsibility to the persistent storage media.

3.1.3.3 Tests

67 **Entire section modified by TD0365**

68 **Test 1:** Applied to each key held as in volatile memory and subject to destruction by overwrite by the TOE (whether or not the value is subsequently encrypted for storage in volatile or non-volatile memory). In the case where the only selection made for the destruction method key was removal of power, then this test is unnecessary. The evaluator will:

1. Record the value of the key in the TOE subject to clearing.
2. Cause the TOE to perform a normal cryptographic processing with the key from Step #1.
3. Cause the TOE to clear the key.
4. Cause the TOE to stop the execution but not exit.
5. Cause the TOE to dump the entire memory of the TOE into a binary file.

6. Search the content of the binary file created in Step #5 for instances of the known key value from Step #1.

69 Steps 1-6 ensure that the complete key does not exist anywhere in volatile memory. If a copy is found, then the test fails.

Findings:	TOE has claimed that removal of power will result in destruction of keys in volatile memory. Therefore, this test is not applicable.

70 **Test 2:** Applied to each key help in non-volatile memory and subject to destruction by the TOE. The evaluator will use special tools (as needed), provided by the TOE developer if necessary, to ensure the tests function as intended.

1. Identify the purpose of the key and what access should fail when it is deleted. (e.g. the data encryption key being deleted would cause data decryption to fail.)

2. Cause the TOE to clear the key.

3. Have the TOE attempt the functionality that the cleared key would be necessary for.

71 The test succeeds if step 3 fails.

High-Level Test Description
Construct an encrypted ZFS dataset using a KEK which is stored in a file. Store an SSH host private key on the encrypted ZFS filesystem. Run an instance of the SSH daemon which uses a host-key on the encrypted dataset. Unmount the ZFS dataset. Destroy the KEK file and attempt to remount the encrypted dataset (it will fail). Rerun the SSH daemon instance and show it fails to load due to a missing hostkey (due to a missing mount).
PASS

72 Tests 3 and 4 do not apply for the selection instructing the underlying platform to destroy the representation of the key, as the TOE has no visibility into the inner workings and completely relies on the underlying platform.

73 **Test 3**: The following tests are used to determine the TOE is able to request the platform to overwrite the key with a TOE supplied pattern.

74 Applied to each key held in non-volatile memory and subject to destruction by overwrite by the TOE. The evaluator will use a tool that provides a logical view of the media (e.g., MBR file system):

1. Record the value of the key in the TOE subject to clearing.

2. Cause the TOE to perform a normal cryptographic processing with the key from Step #1.

3. Cause the TOE to clear the key.

4. Search the logical view that the key was stored in for instances of the known key value from Step #1. If a copy is found, then the test fails.

Findings:	TOE has claimed that that it destroys the abstraction representing the key. This test is therefore not necessary.

75 **Test 4**: Applied to each key held as non-volatile memory and subject to destruction by overwrite by the TOE. The evaluator will use a tool that provides a logical view of the media:

1. Record the logical storage location of the key in the TOE subject to clearing.
2. Cause the TOE to perform a normal cryptographic processing with the key from Step #1.
3. Cause the TOE to clear the key.
4. Read the logical storage location in Step #1 of non-volatile memory to ensure the appropriate pattern is utilized.

76 The test succeeds if correct pattern is used to overwrite the key in the memory location. If the pattern is not found the test fails.

. **Findings:**	TOE has claimed that that it destroys the abstraction representing the key. This test is therefore not necessary.

3.1.4 FCS_COP.1(1) Cryptographic Operation - Encryption/Decryption (Refined)

3.1.4.1 Guidance Documentation

77 The evaluator will verify that the AGD documents contains instructions required to configure the OS to use the required modes and key sizes.

Findings:	The [SUPP], in section 3.4.1, provides guidance to the administrators about the appropriate settings to configure the SSH server. The appropriate ciphers and key sizes are provided. Configuration of the overall SSH server is provided as a link to the sshd(8) man page. For TLS, the permitted ciphers are listed in section 3.4.2 of the [SUPP]. In addition, developers are provided an appropriate link the OpenSSL man pages which detail the means by which the compliant ciphers can be configured.

3.1.4.2 Tests

78 The evaluator will execute all instructions as specified to configure the OS to the appropriate state. The evaluator will perform all of the following tests for each algorithm implemented by the OS and used to satisfy the requirements of this PP:

AES-CBC Known Answer Tests

79 There are four Known Answer Tests (KATs), described below. In all KATs, the plaintext, ciphertext, and IV values shall be 128-bit blocks. The results from each test may either be obtained by the evaluator directly or by supplying the inputs to the implementer and receiving the results in response. To determine correctness, the evaluator will compare the resulting values to those obtained by submitting the same inputs to a known good implementation.

80 KAT-1. To test the encrypt functionality of AES-CBC, the evaluator will supply a set of 10 plaintext values and obtain the ciphertext value that results from AES-CBC encryption of the given plaintext using a key value of all zeros and an IV of all zeros. Five plaintext values shall be encrypted with a 128-bit all-zeros key, and the other five shall be encrypted with a 256-bit all- zeros key. To test the decrypt functionality of AES-CBC, the evaluator will perform the same test as for encrypt, using 10 ciphertext values as input and AES-CBC decryption.

81 KAT-2. To test the encrypt functionality of AES-CBC, the evaluator will supply a set of 10 key values and obtain the ciphertext value that results from AES-CBC encryption of an all-zeros plaintext using the given key value and an IV of all zeros. Five of the keys shall be 128-bit keys, and the other five shall be 256-bit keys. To test the decrypt functionality of AES-CBC, the evaluator will perform the same test as for encrypt, using an all-zero ciphertext value as input and AES-CBC decryption.

82 KAT-3. To test the encrypt functionality of AES-CBC, the evaluator will supply the two sets of key values described below and obtain the ciphertext value that results from AES encryption of an all-zeros plaintext using the given key value and an IV of all zeros. The first set of keys shall have 128 128-bit keys, and the second set shall have 256 256-bit keys. Key i in each set shall have the leftmost i bits be ones and the rightmost N-i bits be zeros, for i in [1,N]. To test the decrypt functionality of AES-CBC, the evaluator will supply the two sets of key and ciphertext value pairs described below and obtain the plaintext value that results from AES-CBC decryption of the given ciphertext using the given key and an IV of all zeros. The first set of key/ciphertext pairs shall have 128 128-bit key/ciphertext pairs, and the second set of key/ciphertext pairs shall have 256 256-bit key/ciphertext pairs. Key i in each set shall have the leftmost i bits be ones and the rightmost N-i bits be zeros, for i in [1,N]. The ciphertext value in each pair shall be the value that results in an all-zeros plaintext when decrypted with its corresponding key.

83 KAT-4. To test the encrypt functionality of AES-CBC, the evaluator will supply the set of 128 plaintext values described below and obtain the two ciphertext values that result from AES-CBC encryption of the given plaintext using a 128-bit key value of all zeros with an IV of all zeros and using a 256-bit key value of all zeros with an IV of all zeros, respectively. Plaintext value i in each set shall have the leftmost i bits be ones and the rightmost 128-i bits be zeros, for i in [1,128].

84 To test the decrypt functionality of AES-CBC, the evaluator will perform the same test as for encrypt, using ciphertext values of the same form as the plaintext in the encrypt test as input and AES-CBC decryption.

AES-CBC Multi-Block Message Test

85 The evaluator will test the encrypt functionality by encrypting an i-block message where $1 < i \leq 10$. The evaluator will choose a key, an IV and plaintext message of length i blocks and encrypt the message, using the mode to be tested, with the chosen key and IV. The ciphertext shall be compared to the result of encrypting the same plaintext message with the same key and IV using a known good implementation. The evaluator will also test the decrypt functionality for each mode by decrypting an i-block message where $1 < i \leq 10$. The evaluator will choose a key, an IV and a ciphertext message of length i blocks and decrypt the message, using the mode to be tested, with the chosen key and IV. The plaintext shall be compared to the result of decrypting the same ciphertext message with the same key and IV using a known good implementation.

AES-CBC Monte Carlo Tests

86 The evaluator will test the encrypt functionality using a set of 200 plaintext, IV, and key 3- tuples. 100 of these shall use 128 bit keys, and 100 shall use 256 bit keys. The

plaintext and IV values shall be 128-bit blocks. For each 3-tuple, 1000 iterations shall be run as follows:

```
# Input: PT, IV, Key
for i = 1 to 1000:
  if i == 1:
      CT[1] = AES-CBC-Encrypt(Key, IV, PT)
      PT = IV
  else:
    CT[i] = AES-CBC-Encrypt(Key, PT)
    PT = CT[i-1]
```

87 The ciphertext computed in the 1000th iteration (i.e., CT[1000]) is the result for that trial. This result shall be compared to the result of running 1000 iterations with the same values using a known good implementation.

88 The evaluator will test the decrypt functionality using the same test as for encrypt, exchanging CT and PT and replacing AES-CBC-Encrypt with AES-CBC-Decrypt.

Findings:	AES-CBC mode with 128-bit and 256-bit keys is claimed for SSH and TLS functionality. CAVP C1651 has the appropriate certificates for the claimed platforms for this mode and key sizes for both encrypt and decrypt: https://csrc.nist.gov/projects/cryptographic-algorithm-validation-program/details?product=12239

AES-GCM Monte Carlo Tests

89 The evaluator will test the authenticated encrypt functionality of AES-GCM for each combination of the following input parameter lengths:

90 128 bit and 256 bit keys

91 Two plaintext lengths. One of the plaintext lengths shall be a non-zero integer multiple of 128 bits, if supported. The other plaintext length shall not be an integer multiple of 128 bits, if supported.

92 Three AAD lengths. One AAD length shall be 0, if supported. One AAD length shall be a non-zero integer multiple of 128 bits, if supported. One AAD length shall not be an integer multiple of 128 bits, if supported.

93 Two IV lengths. If 96 bit IV is supported, 96 bits shall be one of the two IV lengths tested.

94 The evaluator will test the encrypt functionality using a set of 10 key, plaintext, AAD, and IV tuples for each combination of parameter lengths above and obtain the ciphertext value and tag that results from AES-GCM authenticated encrypt. Each supported tag length shall be tested at least once per set of 10. The IV value may be supplied by the evaluator or the implementation being tested, as long as it is known.

95 The evaluator will test the decrypt functionality using a set of 10 key, ciphertext, tag, AAD, and IV 5-tuples for each combination of parameter lengths above and obtain a Pass/Fail result on authentication and the decrypted plaintext if Pass. The set shall include five tuples that Pass and five that Fail.

96 The results from each test may either be obtained by the evaluator directly or by supplying the inputs to the implementer and receiving the results in response. To determine correctness, the evaluator will compare the resulting values to those obtained by submitting the same inputs to a known good implementation.

Findings:	AES-GCM mode with 128-bit and 256-bit keys is claimed for SSH and TLS functionality and for use in ZFS encrypted volumes. CAVP C1651 has the appropriate certificates for the claimed platforms for this mode and key sizes for both encrypt and decrypt: https://csrc.nist.gov/projects/cryptographic-algorithm-validation-program/details?product=12239

AES-CCM Tests

97 The evaluator will test the generation-encryption and decryption-verification functionality of AES-CCM for the following input parameter and tag lengths:

98 128 bit and 256 bit keys

99 Two payload lengths. One payload length shall be the shortest supported payload length, greater than or equal to zero bytes. The other payload length shall be the longest supported payload length, less than or equal to 32 bytes (256 bits).

100 Two or three associated data lengths. One associated data length shall be 0, if supported. One associated data length shall be the shortest supported payload length, greater than or equal to zero bytes. One associated data length shall be the longest supported payload length, less than or equal to 32 bytes (256 bits). If the implementation supports an associated data length of 2 16 bytes, an associated data length of 216 bytes shall be tested.

101 Nonce lengths. All supported nonce lengths between 7 and 13 bytes, inclusive, shall be tested.

102 Tag lengths. All supported tag lengths of 4, 6, 8, 10, 12, 14 and 16 bytes shall be tested.

103 To test the generation-encryption functionality of AES-CCM, the evaluator will perform the following four tests:

104 Test 1: For EACH supported key and associated data length and ANY supported payload, nonce and tag length, the evaluator will supply one key value, one nonce value and 10 pairs of associated data and payload values and obtain the resulting ciphertext.

105 Test 2: For EACH supported key and payload length and ANY supported associated data, nonce and tag length, the evaluator will supply one key value, one nonce value and 10 pairs of associated data and payload values and obtain the resulting ciphertext.

106 Test 3: For EACH supported key and nonce length and ANY supported associated data, payload and tag length, the evaluator will supply one key value and 10 associated data, payload and nonce value 3-tuples and obtain the resulting ciphertext.

107 Test 4: For EACH supported key and tag length and ANY supported associated data, payload and nonce length, the evaluator will supply one key value, one nonce value and 10 pairs of associated data and payload values and obtain the resulting ciphertext.

108 To determine correctness in each of the above tests, the evaluator will compare the ciphertext with the result of generation-encryption of the same inputs with a known good implementation.

109 To test the decryption-verification functionality of AES-CCM, for EACH combination of supported associated data length, payload length, nonce length and tag length, the evaluator shall supply a key value and 15 nonce, associated data and ciphertext 3-tuples and obtain either a FAIL result or a PASS result with the decrypted payload. The evaluator will supply 10 tuples that should FAIL and 5 that should PASS per set of 15.

110 Additionally, the evaluator will use tests from the IEEE 802.11-02/362r6 document "Proposed Test vectors for IEEE 802.11 TGi", dated September 10, 2002, Section 2.1 AESCCMP Encapsulation Example and Section 2.2 Additional AES CCMP Test Vectors to further verify the IEEE 802.11-2007 implementation of AES-CCMP.

Findings:	AES-CCM mode with 128-bit and 256-bit keys is claimed for ZFS encrypted volumes. CAVP C1651 has the appropriate certificates for the claimed platforms for this mode and key sizes: https://csrc.nist.gov/projects/cryptographic-algorithm-validation-program/details?product=12239. In addition, CAVP C1895 (kernel cryptographic framework) also has AES-CCM for both 128- and 256-bit keys: https://csrc.nist.gov/projects/cryptographic-algorithm-validation-program/details?product=12702

AES-GCM Test

111 The evaluator will test the authenticated encrypt functionality of AES-GCM for each combination of the following input parameter lengths:

112 128 bit and 256 bit keys

113 Two plaintext lengths. One of the plaintext lengths shall be a non-zero integer multiple of 128 bits, if supported. The other plaintext length shall not be an integer multiple of 128 bits, if supported.

114 Three AAD lengths. One AAD length shall be 0, if supported. One AAD length shall be a non-zero integer multiple of 128 bits, if supported. One AAD length shall not be an integer multiple of 128 bits, if supported.

115 Two IV lengths. If 96 bit IV is supported, 96 bits shall be one of the two IV lengths tested.

116 The evaluator will test the encrypt functionality using a set of 10 key, plaintext, AAD, and IV tuples for each combination of parameter lengths above and obtain the ciphertext value and tag that results from AES-GCM authenticated encrypt. Each supported tag length shall be tested at least once per set of 10. The IV value may be supplied by the evaluator or the implementation being tested, as long as it is known.

117 The evaluator will test the decrypt functionality using a set of 10 key, ciphertext, tag, AAD, and IV 5-tuples for each combination of parameter lengths above and obtain a

Pass/Fail result on authentication and the decrypted plaintext if Pass. The set shall include five tuples that Pass and five that Fail.

118 The results from each test may either be obtained by the evaluator directly or by supplying the inputs to the implementer and receiving the results in response. To determine correctness, the evaluator will compare the resulting values to those obtained by submitting the same inputs to a known good implementation.

Findings:	See AES-GCM above.

XTS-AES Test

119 The evaluator will test the encrypt functionality of XTS-AES for each combination of the following input parameter lengths:

120 256 bit (for AES-128) and 512 bit (for AES-256) keys

121 Three data unit (i.e., plaintext) lengths. One of the data unit lengths shall be a nonzero integer multiple of 128 bits, if supported. One of the data unit lengths shall be an integer multiple of 128 bits, if supported. The third data unit length shall be either the longest supported data unit length or 216 bits, whichever is smaller.

122 using a set of 100 (key, plaintext and 128-bit random tweak value) 3-tuples and obtain the ciphertext that results from XTS-AES encrypt.

123 The evaluator may supply a data unit sequence number instead of the tweak value if the implementation supports it. The data unit sequence number is a base-10 number ranging between 0 and 255 that implementations convert to a tweak value internally.

124 The evaluator will test the decrypt functionality of XTS-AES using the same test as for encrypt, replacing plaintext values with ciphertext values and XTS-AES encrypt with XTSAES decrypt.

Findings:	AES-XTS is not a claimed mode of AES in the TOE.

AES Key Wrap (AES-KW) and Key Wrap with Padding (AES-KWP) Test

125 The evaluator will test the authenticated encryption functionality of AES-KW for EACH combination of the following input parameter lengths:

126 128 and 256 bit key encryption keys (KEKs)

127 Three plaintext lengths. One of the plaintext lengths shall be two semi-blocks (128 bits). One of the plaintext lengths shall be three semi-blocks (192 bits). The third data unit length shall be the longest supported plaintext length less than or equal to 64 semi-blocks (4096 bits).

128 using a set of 100 key and plaintext pairs and obtain the ciphertext that results from AES-KW authenticated encryption. To determine correctness, the evaluator will use the AES-KW authenticated-encryption function of a known good implementation.

129 The evaluator will test the authenticated-decryption functionality of AES-KW using the same test as for authenticated-encryption, replacing plaintext values with ciphertext values and AES-KW authenticated-encryption with AES-KW authenticated-decryption.

130 The evaluator will test the authenticated-encryption functionality of AES-KWP using the same test as for AES-KW authenticated-encryption with the following change in the three plaintext lengths:

131 One plaintext length shall be one octet. One plaintext length shall be 20 octets (160 bits).

132 One plaintext length shall be the longest supported plaintext length less than or equal to 512 octets (4096 bits).

133 The evaluator will test the authenticated-decryption functionality of AES-KWP using the same test as for AES-KWP authenticated-encryption, replacing plaintext values with ciphertext values and AES-KWP authenticated-encryption with AES-KWP authenticated-decryption.

Findings:	AES-KW and AES-KWP are not claimed modes of AES in the TOE.

3.1.5 FCS_COP.1(2) Cryptographic Operation - Hashing (Refined)

3.1.5.1 TSS

134 The evaluator will check that the association of the hash function with other application cryptographic functions (for example, the digital signature verification function) is documented in the TSS.

Findings:	Section 6.2.4 of the [ST] specifies the hash function is used for in TLS, SSH and Digital Signatures. The evaluator verified that section 6.2.5 of the [ST] covers Digital Signatures, section 6.2.7 of the [ST] covers SSH, and section 6.2.9 of the [ST] covers TLS. The evaluator verified that the cryptographic functions are documented in the TSS.

3.1.5.2 Tests

135 The TSF hashing functions can be implemented in one of two modes. The first mode is the byte-oriented mode. In this mode the TSF only hashes messages that are an integral number of bytes in length; i.e., the length (in bits) of the message to be hashed is divisible by 8. The second mode is the bit-oriented mode. In this mode the TSF hashes messages of arbitrary length. As there are different tests for each mode, an indication is given in the following sections for the bit-oriented vs. the byte-oriented testmacs. The evaluator will perform all of the following tests for each hash algorithm implemented by the TSF and used to satisfy the requirements of this PP.

136 The following tests require the developer to provide access to a test application that provides the evaluator with tools that are typically not found in the production application.

137 Test 1: Short Messages Test (Bit oriented Mode) - The evaluator will generate an input set consisting of m+1 messages, where m is the block length of the hash algorithm. The length of the messages range sequentially from 0 to m bits. The message text shall be pseudorandomly generated. The evaluator will compute the message digest for each of the messages and ensure that the correct result is produced when the messages are provided to the TSF.

138 Test 2: Short Messages Test (Byte oriented Mode) - The evaluator will generate an input set consisting of m/8+1 messages, where m is the block length of the hash algorithm. The length of the messages range sequentially from 0 to m/8 bytes, with

each message being an integral number of bytes. The message text shall be pseudorandomly generated. The evaluator will compute the message digest for each of the messages and ensure that the correct result is produced when the messages are provided to the TSF.

139 Test 3: Selected Long Messages Test (Bit oriented Mode) - The evaluator will generate an input set consisting of m messages, where m is the block length of the hash algorithm. The length of the ith message is 512 + 99·i, where 1 ≤ i ≤ m. The message text shall be pseudorandomly generated. The evaluator will compute the message digest for each of the messages and ensure that the correct result is produced when the messages are provided to the TSF.

140 Test 4: Selected Long Messages Test (Byte oriented Mode) - The evaluator will generate an input set consisting of m/8 messages, where m is the block length of the hash algorithm. The length of the ith message is 512 + 8·99·i, where 1 ≤ i ≤ m/8. The message text shall be pseudorandomly generated. The evaluator will compute the message digest for each of the messages and ensure that the correct result is produced when the messages are provided to the TSF.

141 Test 5: Pseudorandomly Generated Messages Test - This test is for byte-oriented implementations only. The evaluator will randomly generate a seed that is n bits long, where n is the length of the message digest produced by the hash function to be tested. The evaluator will then formulate a set of 100 messages and associated digests by following the algorithm provided in Figure 1 of [SHAVS]. The evaluator will then ensure that the correct result is produced when the messages are provided to the TSF.

Findings:	SSH and TLS use of hashing is provided by CAVP C1651: https://csrc.nist.gov/projects/cryptographic-algorithm-validation-program/details?product=12239. C1651 is also used for digital signatures of OS updates and application software packages. The kernel makes use of RSA digital signatures for verified boot in C1835 and C1937: https://csrc.nist.gov/projects/cryptographic-algorithm-validation-program/details?product=12702 https://csrc.nist.gov/projects/cryptographic-algorithm-validation-program/details?product=12745

3.1.6 FCS_COP.1(3) Cryptographic Operation - Signing (Refined)

3.1.6.1 Tests

142 The evaluator will perform the following activities based on the selections in the ST.

143 The following tests require the developer to provide access to a test application that provides the evaluator with tools that are typically not found in the production application.

ECDSA Algorithm Tests

144 Test 1: ECDSA FIPS 186-4 Signature Generation Test. For each supported NIST curve (i.e., P-256, P-384 and P-521) and SHA function pair, the evaluator will generate 10 1024-bit long messages and obtain for each message a public key and the resulting signature values R and S. To determine correctness, the evaluator will use the signature verification function of a known good implementation.

145 Test 2: ECDSA FIPS 186-4 Signature Verification Test. For each supported NIST curve (i.e., P-256, P-384 and P-521) and SHA function pair, the evaluator will generate a set of 10 1024-bit message, public key and signature tuples and modify one of the values (message, public key or signature) in five of the 10 tuples. The evaluator will verify that 5 responses indicate success and 5 responses indicate failure.

Findings:	The TOE makes use of ECDSA signing for SSH. These services are provided by C1651: https://csrc.nist.gov/projects/cryptographic-algorithm-validation-program/details?product=12239.

RSA Signature Algorithm Tests

146 Test 1: Signature Generation Test. The evaluator will verify the implementation of RSA Signature Generation by the OS using the Signature Generation Test. To conduct this test the evaluator must generate or obtain 10 messages from a trusted reference implementation for each modulus size/SHA combination supported by the TSF. The evaluator will have the OS use its private key and modulus value to sign these messages. The evaluator will verify the correctness of the TSF's signature using a known good implementation and the associated public keys to verify the signatures.

147 Test 2: Signature Verification Test. The evaluator will perform the Signature Verification test to verify the ability of the OS to recognize another party's valid and invalid signatures. The evaluator will inject errors into the test vectors produced during the Signature Verification Test by introducing errors in some of the public keys, e, messages, IR format, and/or signatures. The evaluator will verify that the OS returns failure when validating each signature.

Findings:	The TOE makes use of RSA signing for TLS, SSH, verified boot and trusted updates for OS and application software. These services are provided by C1651: https://csrc.nist.gov/projects/cryptographic-algorithm-validation-program/details?product=12239.

3.1.7 FCS_COP.1(4) Cryptographic Operation - Keyed-Hash Message Authentication (Refined)

3.1.7.1 Tests

148 The evaluator will perform the following activities based on the selections in the ST.

149 For each of the supported parameter sets, the evaluator will compose 15 sets of test data. Each set shall consist of a key and message data. The evaluator will have the OS generate HMAC tags for these sets of test data. The resulting MAC tags shall be compared against the result of generating HMAC tags with the same key and IV using a known-good implementation.

Findings:	The TOE makes use of HMAC for TLS and SSH. These services are provided by C1651: https://csrc.nist.gov/projects/cryptographic-algorithm-validation-program/details?product=12239.

3.1.8 FCS_RBG_EXT.1 Random Bit Generation

FCS_RBG_EXT.1.1

3.1.8.1 TSS

150 Application Note: NIST SP 800-90A contains three different methods of generating random numbers; each of these, in turn, depends on underlying cryptographic primitives (hash functions/ciphers). The ST author will select the function used and include the specific underlying cryptographic primitives used in the requirement or in the TSS.

Findings:	The TOE claims two distinct DRBGs. Both are Hash_DRBG using a SHA2-512 hash function. The SHA2-512 hash function is claimed in FCS_COP.1(2) in the ST. This hash function is also claimed in CAVP

3.1.8.2 Tests

151 The evaluator will perform the following tests:

152 The evaluator will perform 15 trials for the RNG implementation. If the RNG is configurable, the evaluator will perform 15 trials for each configuration. The evaluator will also confirm that the operational guidance contains appropriate instructions for configuring the RNG functionality.

153 If the RNG has prediction resistance enabled, each trial consists of (1) instantiate DRBG, (2) generate the first block of random bits (3) generate a second block of random bits (4) uninstantiate. The evaluator verifies that the second block of random bits is the expected value. The evaluator will generate eight input values for each trial. The first is a count (0 – 14). The next three are entropy input, nonce, and personalization string for the instantiate operation. The next two are additional input and entropy input for the first call to generate. The final two are additional input and entropy input for the second call to generate. These values are randomly generated. "generate one block of random bits" means to generate random bits with number of returned bits equal to the Output Block Length (as defined in NIST SP 800-90A).

154 If the RNG does not have prediction resistance, each trial consists of (1) instantiate DRBG, (2) generate the first block of random bits (3) reseed, (4) generate a second block of random bits (5) uninstantiate. The evaluator verifies that the second block of random bits is the expected value. The evaluator will generate eight input values for each trial. The first is a count (0 – 14). The next three are entropy input, nonce, and personalization string for the instantiate operation. The fifth value is additional input to the first call to generate. The sixth and seventh are additional input and entropy input to the call to reseed. The final value is additional input to the second generate call.

155 The following list contains more information on some of the input values to be generated/selected by the evaluator.

1. Entropy input: The length of the entropy input value must equal the seed length.

2. Nonce: If a nonce is supported (CTR_DRBG with no Derivation Function does not use a nonce), the nonce bit length is one-half the seed length.

3. Personalization string: The length of the personalization string must be less than or equal to seed length. If the implementation only supports one personalization string length, then the same length can be used for both values. If more than one string length is support, the evaluator will use personalization strings of two different lengths. If the implementation does not use a personalization string, no value needs to be supplied.

4. Additional input: The additional input bit lengths have the same defaults and restrictions as the personalization string lengths.

Findings:	The DRBGs are included in C1651 (https://csrc.nist.gov/projects/cryptographic-algorithm-validation-program/details?product=12239) and C1895 (https://csrc.nist.gov/projects/cryptographic-algorithm-validation-program/details?product=12702).

FCS_RBG_EXT.1.2

156 Documentation shall be produced - and the evaluator will perform the activities - in accordance with Appendix E - Entropy Documentation and Assessment and the Clarification to the Entropy Documentation and Assessment Annex.

157 In the future, specific statistical testing (in line with NIST SP 800-90B) will be required to verify the entropy estimates.

Findings:	A standalone entropy document has been submitted to the CB for review and assessment. It was found to meet the requirements laid out in Appendix E.

3.1.9 FCS_STO_EXT.1 Storage of Sensitive Data

3.1.9.1 TSS

158 The evaluator will check the TSS to ensure that it lists all persistent sensitive data for which the OS provides a storage capability. For each of these items, the evaluator will confirm that the TSS lists for what purpose it can be used, and how it is stored.

Findings:	Section 6.2.2 of the [ST] includes a table of persistent sensitive data. This list includes the purpose as specified by the name of the key, and how the key is stored. Section 6.2.8 of the [ST] specifies that the TOE uses AES-GCM-128/256 or AES-CCM-128/256 to protect sensitive data.

159 The evaluator will confirm that cryptographic operations used to protect the data occur as specified in FCS_COP.1(1).

Findings:	SSH host keys are stored on a ZFS encrypted volume which means they are encrypted using AES-CCM or AES-GCM. The ZFS DEK is encrypted using a ZFS KEK which means they are encrypted using AES-CCM or AES-GCM. Both of these algorithms are claimed in FCS_COP.1(1).

160 Application Note: Sensitive data shall be identified in the TSS by the ST author, and minimally includes credentials and keys.

Findings:	In section 6.2.8 of the [ST], the author broadly claims that all data and file system metadata are encrypted when stored using ZFS. This implies that the term “sensitive” data is implicitly defined as anything considered by the end-administrator needing to be stored in an encrypted dataset (in addition to credentials). Local credentials are cryptographically hashed in /etc/shadow(5) using SHA256 by default. Hashing algorithms can be selected in /etc/security/crypt.conf as per the man page for crypto.conf(5)

3.1.9.2 Guidance Documentation

161 The evaluator will also consult the developer documentation to verify that an interface exists for applications to securely store credentials.

Findings:	In the [SUPP] section 3.3.2 the /etc/shadow is the mechanism to securely store credentials. The passwd CLI program is the means by which users are permitted to interact with this credential store.

3.1.10 FCS_TLSC_EXT.1 TLS Client Protocol

FCS_TLSC_EXT.1.1

3.1.10.1 TSS

162 The evaluator will check the description of the implementation of this protocol in the TSS to ensure that the cipher suites supported are specified. The evaluator will check the TSS to ensure that the cipher suites specified include those listed for this component.

Findings:	Section 6.2.9 of the [ST] specifies the cipher suites supported. The list in section 6.2.9 of the [ST] matches the selections in the associated SFR.

3.1.10.2 Guidance Documentation

163 The evaluator will also check the operational guidance to ensure that it contains instructions on configuring the OS so that TLS conforms to the description in the TSS.

Findings:	The TLS ciphersuites are described in section 3.4.2 of [SUPP]. The reader is directed to the various man pages for OpenSSL as to how to configure the ciphersuites for use. By restricting the set of ciphersuites to those provided in the [SUPP] section 3.4.2, a Solaris developer will ensure that the TOE conforms with the description in the TSS.

3.1.10.3 Tests

164 The evaluator will also perform the following tests:

165 Test 1: The evaluator will establish a TLS connection using each of the cipher suites specified by the requirement. This connection may be established as part of the establishment of a higher-level protocol, e.g., as part of an EAP session. It is sufficient to observe the successful negotiation of a cipher suite to satisfy the intent of the test;

it is not necessary to examine the characteristics of the encrypted traffic in an attempt to discern the cipher suite being used (for example, that the cryptographic algorithm is 128-bit AES and not 256-bit AES).

High-Level Test Description
Using a TLS client, connect to the TLS server in the environment and show that the claimed ciphersuites are supported. As per FCS_CKM.2 TD501 test AA update, all claimed safe primes are tested and shown to work.
PASS

166 Test 2: The evaluator will attempt to establish the connection using a server with a server certificate that contains the Server Authentication purpose in the extendedKeyUsage field and verify that a connection is established. The evaluator will then verify that the client rejects an otherwise valid server certificate that lacks the Server Authentication purpose in the extendedKeyUsage field and a connection is not established. Ideally, the two certificates should be identical except for the extendedKeyUsage field.

High-Level Test Description
Using the TOE TLS client, show that when connecting to the TLS server in the environment which is missing the serverAuth property in the extendedKeyUsage extension, that the TLS client will terminate.
PASS

167 Test 3: The evaluator will send a server certificate in the TLS connection that does not match the server-selected cipher suite (for example, send a ECDSA certificate while using the TLS_RSA_WITH_AES_128_CBC_SHA cipher suite or send a RSA certificate while using one of the ECDSA cipher suites.) The evaluator will verify that the OS disconnects after receiving the server's Certificate handshake message.

High-Level Test Description
Using a TOE TLS client, connect to a TLS server in the environment which sends back a certificate which does not match the negotiated ciphersuite and show that the connection is terminated.
PASS

168 Test 4: The evaluator will configure the server to select the TLS_NULL_WITH_NULL_NULL cipher suite and verify that the client denies the connection.

High-Level Test Description
Using a TLS client, connect to the TLS server in the environment using the TLS_NULL_WITH_NULL_NULL cipher and show that the connection is denied.
PASS

169 Test 5: The evaluator will perform the following modifications to the traffic:

Test 5.1: Change the TLS version selected by the server in the Server Hello to a non-supported TLS version (for example 1.3 represented by the two bytes 03 04) and verify that the client rejects the connection.

High-Level Test Description
Connect to a TLS server in the environment and show that the TOE TLS client fails to connect when presented with an incorrect protocol version string.
PASS

Test 5.2: Modify at least one byte in the server's nonce in the Server Hello handshake message, and verify that the client rejects the Server Key Exchange handshake message (if using a DHE or ECDHE cipher suite) or that the server denies the client's Finished handshake message.

High-Level Test Description
Using the TOE TLS client, connect to the TLS server in the environment and show that the client rejects the connection when the server key exchange message has been modified as per the test case. Do this for both RSA- and DHE-based key exchange ciphersuites.
PASS

Test 5.3: Modify the server's selected cipher suite in the Server Hello handshake message to be a cipher suite not presented in the Client Hello handshake message. The evaluator will verify that the client rejects the connection after receiving the Server Hello.

High-Level Test Description
Using a TLS client, connect to the TLS server in the environment which will attempt to negotiate a ciphersuite not presented in the TOE's Client Hello ciphersuite list. Show the connection is denied..
PASS

Test 5.4: If an ECDHE or DHE ciphersuite is selected, modify the signature block in the Server's Key Exchange handshake message, and verify that the client rejects the connection after receiving the Server Key Exchange message.

High-Level Test Description
Using the TOE TLS client, connect to a TLS server in the environment which will modify the Server Key Exchange handshaking message signature and verify the connection fails.
PASS

Test 5.5: Modify a byte in the Server Finished handshake message, and verify that the client sends a fatal alert upon receipt and does not send any application data.

High-Level Test Description
Using the TOE TLS client, connect to a TLS server in the environment which will modify the Finished handshaking message and verify the connection does not send any Application Data.

High-Level Test Description
PASS

Test 5.6: Send a garbled message from the Server after the Server has issued the Change Cipher Spec message and verify that the client denies the connection.

High-Level Test Description
Using the TOE TLS client, connect to a TLS server in the environment which will send back a garbled message after the ChangeCipherSpec has been issued and show the connection fails.
PASS

FCS_TLSC_EXT.1.2

3.1.10.4 TSS

170 The evaluator will ensure that the TSS describes the client's method of establishing all reference identifiers from the application-configured reference identifier, including which types of reference identifiers are supported (e.g. Common Name, DNS Name, URI Name, Service Name, or other application-specific Subject Alternative Names) and whether IP addresses and wildcards are supported. The evaluator will ensure that this description identifies whether and the manner in which certificate pinning is supported or used by the OS.

Findings:	In section 6.2.9 of the [ST], the TSS indicates that TLS clients are required to use established OpenSSL API mechanisms to set the reference identifiers. The TOE supports IP addresses and DNS names. Wildcards are supported. Certificate pinning is not supported.

3.1.10.5 Guidance Documentation

171 The evaluator will verify that the AGD guidance includes instructions for setting the reference identifier to be used for the purposes of certificate validation in TLS.

Findings:	Section 3.4.2 of the [SUPP] provides the appropriate OpenSSL API mechanisms to set the hostname and IP addresses.

3.1.10.6 Tests

172 The evaluator will configure the reference identifier according to the AGD guidance and perform the following tests during a TLS connection:

173 Test 1: The evaluator will present a server certificate that does not contain an identifier in either the Subject Alternative Name (SAN) or Common Name (CN) that matches the reference identifier. The evaluator will verify that the connection fails.

High-Level Test Description
Using the TOE TLS client, connect to a TLS server in the environment which will send back a certificate in which the Common Name does not match the reference name. Show the connection will fail. Repeat, but this time use the Subject Alternative Name. Ensure the test case is perform for each of the claimed identifier types.
PASS

174 Test 2: The evaluator will present a server certificate that contains a CN that matches the reference identifier, contains the SAN extension, but does not contain an identifier in the SAN that matches the reference identifier. The evaluator shall verify that the connection fails. The evaluator will repeat this test for each supported SAN type.

High-Level Test Description
Using the TOE TLS client, connect to the TLS server in the environment. The server will return an X.509 certificate which has a CN that matches and a SAN which does not. The TLS client is expected to fail to connect. Repeat for each identifier type.
PASS

175 Test 3: [conditional] If the TOE does not mandate the presence of the SAN extension, the evaluator will present a server certificate that contains a CN that matches the reference identifier and does not contain the SAN extension. The evaluator will verify that the connection succeeds. If the TOE mandates the presence of the SAN extension, this test shall be omitted.

High-Level Test Description
Using the TOE TLS client, connect to the TLS server in the environment. The server will return an X.509 certificate which has a CN that matches and is missing the SAN. The TLS client is expected to succeed. Repeat for each identifier type.
PASS

176 Test 4: The evaluator will present a server certificate that contains a CN that does not match the reference identifier but does contain an identifier in the SAN that matches. The evaluator will verify that the connection succeeds.

High-Level Test Description
Using the TOE TLS client, connect to the TLS server in the environment. The server will return an X.509 certificate which has a CN that does not match and a SAN which matches. The TLS client is expected to succeed. Repeat for each SAN identifier type.
PASS

177 Test 5: The evaluator will perform the following wildcard tests with each supported type of reference identifier:

Test 5.1: The evaluator will present a server certificate containing a wildcard that is not in the left-most label of the presented identifier (e.g. foo.*.example.com) and verify that the connection fails.

High-Level Test Description
Using the TOE TLS client, connect to the TLS server in the environment. The server will return an X.509 certificate which has an invalid wildcard as described in the test case. The connection will fail. The test will be repeated for a similar wildcard in the SAN extension.
PASS

Test 5.2: The evaluator will present a server certificate containing a wildcard in the left-most label but not preceding the public suffix (e.g. *.example.com). The evaluator will configure the reference identifier with a single left-most label (e.g. foo.example.com) and verify that the connection succeeds. The evaluator will configure the reference identifier without a left-most label as in the certificate (e.g. example.com) and verify that the connection fails. The evaluator will configure the reference identifier with two left-most labels (e.g. bar.foo.example.com) and verify that the connection fails.

High-Level Test Description
Using the TOE TLS client, connect to the TLS server in the environment. The server will return an X.509 certificate which has a valid wildcard as described in the test case. The connection will succeed or fail as required by the test. The test will be repeated for a similar wildcard in the SAN extension.
PASS

Test 5.3: The evaluator will present a server certificate containing a wildcard in the left-most label immediately preceding the public suffix (e.g. *.com). The evaluator will configure the reference identifier with a single left-most label (e.g. foo.com) and verify that the connection fails. The evaluator will configure the reference identifier with two left-most labels (e.g. bar.foo.com) and verify that the connection fails.

High-Level Test Description
Using the TOE TLS client, connect to the TLS server in the environment. The server will return an X.509 certificate which has an invalid wildcard as described in the test case. The client will attempt several reference identifiers. The connections will fail. The test will be repeated for a similar wildcard in the SAN extension.
PASS

178 Test 6: [conditional] If URI or Service name reference identifiers are supported, the evaluator will configure the DNS name and the service identifier. The evaluator will present a server certificate containing the correct DNS name and service identifier in the URIName or SRVName fields of the SAN and verify that the connection succeeds. The evaluator will repeat this test with the wrong service identifier (but correct DNS name) and verify that the connection fails.

NOTE:	URI and service name types are not claimed.

179 Test 7: [conditional] If pinned certificates are supported the evaluator will present a certificate that does not match the pinned certificate and verify that the connection fails.

NOTE:	Pinned certificates are not claimed.

FCS_TLSC_EXT.1.3

3.1.10.7 Tests

180 The evaluator will use TLS as a function to verify that the validation rules in FIA_X509_EXT.1.1 are adhered to and shall perform the following additional test:

181 Test 1: The evaluator will demonstrate that a peer using a certificate without a valid certification path results in an authenticate failure. Using the administrative guidance, the evaluator will then load the trusted CA certificate(s) needed to validate the peer's certificate, and demonstrate that the connection succeeds. The evaluator then shall delete one of the CA certificates, and show that the connection fails.

NOTE:	Each of these failures are shown in FIA_X509_EXT.1/Rev. In each case the expected action is witnessed.

182 Test 2: The evaluator will demonstrate that a peer using a certificate which has been revoked results in an authentication failure.

NOTE:	Each of these failures are shown in FIA_X509_EXT.1/Rev. In each case the expected action is witnessed.

183 Test 3: The evaluator will demonstrate that a peer using a certificate which has passed its expiration date results in an authentication failure.

NOTE:	Each of these failures are shown in FIA_X509_EXT.1/Rev. In each case the expected action is witnessed.

184 Test 4: the evaluator will demonstrate that a peer using a certificate which does not have a valid identifier shall result in an authentication failure.

NOTE:	Each of these failures are shown in FIA_X509_EXT.1/Rev. In each case the expected action is witnessed.

3.2 User Data Protection (FDP)

3.2.1 FDP_ACF_EXT.1 Access Controls for Protecting User Data

3.2.1.1 TSS

185 The evaluator will confirm that the TSS comprehensively describes the access control policy enforced by the OS. The description must include the rules by which accesses to particular files and directories are determined for particular users. The evaluator will inspect the TSS to ensure that it describes the access control rules in such detail that given any possible scenario between a user and a file governed by the OS the access control decision is unambiguous.

Findings:	Section 6.3.1 of the [ST] specifies that the TOE implements UNIX permissions for access control. This section provides links to online documents which comprehensively describes the access control rules to provide a clear understanding of the interactions between users and files.

3.2.1.2 Tests

186 The evaluator will create two new standard user accounts on the system and conduct the following tests:

187 Test 1: The evaluator will authenticate to the system as the first user and create a file within that user's home directory. The evaluator will then log off the system and log in as the second user. The evaluator will then attempt to read the file created in the first user's home directory. The evaluator will ensure that the read attempt is denied.

High-Level Test Description
After creating two unprivileged users, create a file as user1 in the user's home directory. Then log into the TOE as user2 and attempt to read the contents of the file just created in user1's home directory. The attempt should fail.
PASS

188 Test 2: The evaluator will authenticate to the system as the first user and create a file within that user's home directory. The evaluator will then log off the system and log in as the second user. The evaluator will then attempt to modify the file created in the first user's home directory. The evaluator will ensure that the modification is denied.

High-Level Test Description
Building upon the previous test case, log into the TOE as user2 and attempt to modify the file created by user1 in the first test case. Show this attempt fails.
PASS

189 Test 3: The evaluator will authenticate to the system as the first user and create a file within that user's user directory. The evaluator will then log off the system and log in as the second user. The evaluator will then attempt to delete the file created in the first user's home directory. The evaluator will ensure that the deletion is denied.

High-Level Test Description
Building upon the previous test case, log into the TOE as user2 and attempt to delete the file created by user1 in the first test case. Show this attempt fails.

High-Level Test Description
PASS

190 Test 4: The evaluator will authenticate to the system as the first user. The evaluator will attempt to create a file in the second user's home directory. The evaluator will ensure that the creation of the file is denied.

High-Level Test Description
Building upon the previous test case, log into the TOE as user2 and attempt to create a new file into the home directory of user1. Show this attempt fails.
PASS

191 Test 5: The evaluator will authenticate to the system as the first user and attempt to modify the file created in the first user's home directory. The evaluator will ensure that the modification of the file is accepted.

High-Level Test Description
Building upon the previous test case, log into the TOE as user1 and attempt to modify one of their own files. Show this attempt succeeds.
PASS

192 Test 6: The evaluator will authenticate to the system as the first user and attempt to delete the file created in the first user's directory. The evaluator will ensure that the deletion of the file is accepted.

High-Level Test Description
Building upon the previous test case, log into the TOE as user1 and attempt to delete one of their own files. Show this attempt succeeds.
PASS

3.3 Security management (FMT)

3.3.1 FMT_MOF_EXT.1 Management of security functions behaviour

3.3.1.1 TSS

193 The evaluator will verify that the TSS describes those management functions that are restricted to Administrators, including how the user is prevented from performing those functions, or not able to use any interfaces that allow access to that function.

Findings:	Section 6.5.1 of the [ST] specifies that the management functions are restricted as specified in Section 5.3.5 of the [ST]. Section 6.5.1 of the [ST] further specifies that the TOE maintains privileges to perform administrative actions.

3.3.1.2 Tests

194 **Test 1:** For each function that is indicated as restricted to the administrator, the evaluation shall perform the function as an administrator, as specified in the Operational Guidance, and determine that it has the expected effect as outlined by the Operational Guidance and the SFR. The evaluator will then perform the function (or otherwise attempt to access the function) as a non-administrator and observe that they are unable to invoke that functionality.

High-Level Test Description
For each of the claimed functions, perform the administrative function – as written in the guidance -- as an administrator and show the effect is obtained. Then run each function as a non-privileged user and show the function fails to be invoked. Perform this for all ways in which the management function can be executed.
PASS

3.3.2 FMT_SMF_EXT.1 Specification of Management Functions

3.3.2.1 Guidance Documentation

195 The evaluator will verify that every management function captured in the ST is described in the operational guidance and that the description contains the information required to perform the management duties associated with the management function.

Findings: The [ST] provides the following claims for administrative functionality in FMT_SMF_EXT.1 in section 5.3.5:

- Enable/disable [session timeout]: described in section 3.3.2 of the [SUPP] and requires the administrator to ensure there is an appropriate session timeout control added to global user shell configuration;

- Configure [session] inactivity timeout: similarly described in section 3.3.2 of the [SUPP];

- Configure local audit storage capacity: In [SUPP] section 4.1, there is a link to the Oracle Solaris 11.4 auditing administration guide (“Managing Auditing in Oracle® Solaris 11.4”). Under the subsection “Configuring Local Audit Logs”, there is a section titled “How to Create ZFS File Systems for Audit”. This guide describes the means to set the quota attribute on the underlying ZFS filesystem.

- Configure minimum password Length: Section 3.3.4.3 of [SUPP] describes the SMF service which must be configured to maintain the password complexity requirements.

- Configure minimum number of special characters in password: See above.

- Configure minimum number of numeric characters in password: See above

- Configure minimum number of uppercase characters in password: See above

- Configure minimum number of lowercase characters in password: See above

- Configure lockout policy for unsuccessful authentication attempts through [limiting number of attempts during a time period]: Section 3.3.4.1 of the [SUPP] describe the

	SMF service which must be configured to enable authentication failure limiting and unlocking actions; - Configure host-based firewall: this is described in section 3.2.1 of the [SUPP] - Configure name/address of directory server with which to bind: Within the [SUPP] there is a pointer to the broader Solaris 11.4 information library. In section “Working With Oracle® Solaris 11.4 Directory and Naming Services: LDAP”, information about using the LDAP client is in “Chapter 5: Setting Up LDAP Clients”. Setting up an LDAP client is reasonably straightforward and can be done using the CLI entirely without any configuration files. Once configured, the information is stored in an SMF service. However, configuration continues to use the ldapclient(8) CLI tool. - Configure name/address of audit/logging server to which to send audit/logging records: Within the [SUPP], there is a reference in section 3.3.4.6 to using the audit_remote plugin. Configuring an external ARS server requires Kerberos but the entire process is documented within the Oracle Solaris “Managing Auditing in Oracle® Solaris 11.4” book in chapter 4, under “How to Send Audit Files to a Remote Repository”. - Configure audit rules: Auditing in Solaris 11.4 is a complex task and the [SUPP] is unable to provide a comprehensive treatment of the topic. Instead, the [SUPP], in section 4.1, points the administrator to the auditing administration guide (“Managing Auditing in Oracle® Solaris 11.4”). Auditing rules can be configured system-wide, per user, per role or per-file. The rule syntax is described in all cases and the rule precedence is provided where overlapping or mutually exclusive rules interact. - Configure name/address of network time server: described in section 3.2.4 of the [SUPP].

3.3.2.2 Tests

196 The evaluator will test the OS's ability to provide the management functions by configuring the operating system and testing each option selected from above. The evaluator is expected to test these functions in all the ways in which the ST and guidance documentation state the configuration can be managed.

Note	This test case is covered in FMT_MOF_EXT.1

3.4 Protection of the TSF (FPT)

3.4.1 FPT_ACF_EXT.1Access controls

FPT_ACF_EXT.1.1

3.4.1.1 TSS

197 The evaluator will confirm that the TSS specifies the locations of kernel drivers/modules, security audit logs, shared libraries, system executables, and system configuration files. Every file does not need to be individually identified, but the system's conventions for storing and protecting such files must be specified.

Findings:	Section 6.6.1 of the [ST] specifies the locations of the kernel drivers/modules, security audit logs, shared libraries, system executables, and system configuration files on the TOE.

3.4.1.2 Tests

198 The evaluator will create an unprivileged user account. Using this account, the evaluator will ensure that the following tests result in a negative outcome (i.e., the action results in the OS denying the evaluator permission to complete the action):

- **Test 1**: The evaluator will attempt to modify all kernel drivers and modules.

High-Level Test Description
Using an unprivileged user account, attempt to modify all kernel drivers and modules and show the attempts are unsuccessful.
PASS

- **Test 2**: The evaluator will attempt to modify all security audit logs generated by the logging subsystem.

High-Level Test Description
For each of the files in the defined log and audit set, attempt to modify them as an unprivileged user.
PASS

- **Test 3**: The evaluator will attempt to modify all shared libraries that are used throughout the system.

High-Level Test Description
For each of the files in the defined shared library locations, attempt to modify them as an unprivileged user.
PASS

- **Test 4**: The evaluator will attempt to modify all system executables.

High-Level Test Description
For each of the files in the defined system executables, attempt to modify them as an unprivileged user.
PASS

- **Test 5**: The evaluator will attempt to modify all system configuration files.

High-Level Test Description
For each of the files in the defined system configuration directories, attempt to modify them as an unprivileged user.
PASS

- **Test 6**: The evaluator will attempt to modify any additional components selected.

NOTE:	Not other components were selected.

FPT_ACF_EXT.1.2

3.4.1.3 Tests

199 The evaluator will create an unprivileged user account. Using this account, the evaluator will ensure that the following tests result in a negative outcome (i.e., the action results in the OS denying the evaluator permission to complete the action):

- **Test 1**: The evaluator will attempt to read security audit logs generated by the auditing subsystem

High-Level Test Description
For each of the files defined as auditing and logging files, attempt to read them as an unprivileged user.
PASS

- **Test 2**: The evaluator will attempt to read system-wide credential repositories

High-Level Test Description
For each of the files defined as credential stores, attempt to read them as an unprivileged user.
PASS

- **Test 3**: The evaluator will attempt to read any other object specified in the assignment

NOTE:	Not other components were selected.

3.4.2 FPT_ASLR_EXT.1 Address Space Layout Randomization

3.4.2.1 Tests

200 The evaluator will select 3 executables included with the TSF. If the TSF includes a web browser it must be selected. If the TSF includes a mail client it must be selected. For each of these apps, the evaluator will launch the same executables on two separate instances of the OS on identical hardware and compare all memory mapping locations. The evaluator will ensure that no memory mappings are placed in the same location. If the rare chance occurs that two mappings are the same for a

single executable and not the same for the other two, the evaluator will repeat the test with that executable to verify that in the second test the mappings are different. This test can also be completed on the same hardware and rebooting between application launches.

High-Level Test Description
Select a binary and execute it. Capture a snapshot of the process memory structure. Reboot. Select the same binary and execute it again. Capture the snapshot of the process memory structure. Compare the two memory structures and show that they are different. The TOE does not contain a web browser by default, but it is installable. Therefore, the web browser was installed for this test case even though it is not normally installed. Perform the test using the installed web browser, the SSH service daemon and the remote administration daemon (RAD).
PASS

3.4.3 FPT_SBOP_EXT.1 Stack Buffer Overflow Protection

3.4.3.1 TSS

201 For stack-based OSes, the evaluator will determine that the TSS contains a description of stack-based buffer overflow protections used by the OS. These are referred to by a variety of terms, such as stack cookie, stack guard, and stack canaries.

Findings:	Section 6.6.3 of the [ST] specifies that the TOE uses nxstack to prevent the stacks from being executed.

202 The TSS must include a rationale for any binaries that are not protected in this manner.

Findings:	The description in the TSS implies there are no situations where the stack can be made executable. The TOE contains both 64-bit and 32-bit executable binaries. All 64-bit binaries have a non-executable stack by default. All 32-bit binaries can contain an executable stack. However, the nxstack security extension disables 32-bit stack execution when it is enabled. By default, nxstack is enabled in the TOE and needs to be explicitly disabled by an administrator.

203 For OSes that store parameters/variables separately from control flow values, the evaluator will verify that the TSS describes what data structures control values, parameters, and variables are stored. The evaluator will also ensure that the TSS includes a description of the safeguards that ensure parameters and variables do not intermix with control flow values.

Findings:	The TOE does not store parameters/variables separately from control flow values as per the selection in the relevant SFR.

3.4.3.2 Tests

204 The evaluator will also perform the following test:

205 Test 1: The evaluator will inventory the kernel, libraries, and application binaries to determine those that do not implement stack-based buffer overflow protections. This list should match up with the list provided in the TSS.

High-Level Test Description
The TOE was inventoried for kernel, libraries, and application binaries to determine those that do not implement stack-based buffer overflow protections. This list was found to match up with the list provided in the TSS.
PASS

3.4.4 FPT_TST_EXT.1 Boot Integrity

3.4.4.1 TSS

206 The evaluator will verify that the TSS section of the ST includes a comprehensive description of the boot procedures, including a description of the entire bootchain, for the TSF. The evaluator will ensure that the OS cryptographically verifies each piece of software it loads in the bootchain to include bootloaders and the kernel. Software loaded for execution directly by the platform (e.g. first-stage bootloaders) is out of scope. For each additional category of executable code verified before execution, the evaluator will verify that the description in the TSS describes how that software is cryptographically verified.

Findings:	Section 6.6.4 of the [ST] specifies the firmware verifies and then loads the Oracle Solaris /platform/.../unix module, the initial Oracle Solaris module. In turn, the Oracle Solaris kernel runtime loader krtld, which is part of the unix module, verifies and loads the generic UNIX (genunix) module and subsequent modules.

207 The evaluator will verify that the TSS contains a description of the protection afforded to the mechanism performing the cryptographic verification.

Findings:	Section 6.6.4 of the [ST] specifies that each block of the boot chain verifies the next. The cryptographic mechanism performing the verification is covered by the mechanism.

3.4.4.2 Tests

208 The evaluator will perform the following tests:

209 Test 1: The evaluator will perform actions to cause TSF software to load and observe that the integrity mechanism does not flag any executables as containing integrity errors and that the OS properly boots.

High-Level Test Description
Ensure that the TOE is configured to fail on integrity issues. Reboot the TOE and verify no errors appear.
PASS

210 Test 2: The evaluator will modify a TSF executable that is part of the bootchain verified by the TSF (i.e. Not the first-stage bootloader) and attempt to boot. The evaluator will ensure that an integrity violation is triggered and the OS does not boot

(Care must be taken so that the integrity violation is determined to be the cause of the failure to load the module, and not the fact that in such a way to invalidate the structure of the module.).

High-Level Test Description
Ensure that the TOE is configured to halt on integrity issues. Modify the kernel and reboot the TOE and verify the TOE fails to boot. Show that the TOE halts on boot and that appropriate error messages are displayed on the console.
PASS

211 **TD0493 -** Test 3[conditional]: If the ST author indicates that the integrity verification is performed using a public key in an X509 certificate, the evaluator will verify that the boot integrity mechanism includes a certificate validation according to FIA_X509_EXT.1 for all certificates in the chain from the certificate used for boot integrity to a certificate in the trust store that are not themselves in the trust store. This means that, for each X509 certificate in this chain that is not a trust store element, the evaluator must ensure that revocation information is available to the TOE during the bootstrap mechanism (before the TOE becomes fully operational).

NOTE:	Integrity verification does not claim X.509 certificates for verification.

3.4.5 FPT_TUD_EXT.1 Trusted Update

FPT_TUD_EXT.1.1

3.4.5.1 Guidance Documentation

212 The evaluator will check for an update using procedures described in the documentation.

Findings:	Guidance, best practice and appropriate commands regarding updating the OS through IPS repositories are provided in the [SUPP] in section 2.4.

3.4.5.2 Tests

213 **This entire section modified by TD0463**

214 The evaluator will check for an update using procedures described in the documentation and verify that the OS provides a list of available updates. Testing this capability may require installing and temporarily placing the system into a configuration in conflict with secure configuration guidance which specifies automatic update.

215 The evaluator is also to ensure that the response to this query is authentic by using a digital signature scheme specified in FCS_COP.1(3). The digital signature verification may be performed as part of a network protocol described in FTP_ITC_EXT.1.) If the signature verification is not performed as part of a trusted channel, the evaluator shall send a query response with a bad signature and verify

that the signature verification fails. The evaluator shall then send a query response with a good signature and verify that the signature verification is successful.

High-Level Test Description
Using a network-based package repository, query for updates over an HTTPS connection. This connection is not claimed as a trusted channel in FTP_ITC_EXT.1 and therefore the test case will also verify the digital signature of the connection associated with the package query action.
PASS

FPT_TUD_EXT.1.2

3.4.5.3 TSS

216 The download could originate from the vendor's website, an enterprise-hosted update repository, or another system (e.g. network peer). All supported origins for the update must be indicated in the TSS and evaluated.

Findings:	Section 6.6.5 of the [ST] specifies that the TOE software is distributed from IPS package repositories.

3.4.5.4 Tests

217 For the following tests, the evaluator will initiate the download of an update and capture the update prior to installation. The download could originate from the vendor's website, an enterprise-hosted update repository, or another system (e.g. network peer). All supported origins for the update must be indicated in the TSS and evaluated.

218 Test 1: The evaluator will ensure that the update has a digital signature belonging to the vendor prior to its installation. The evaluator will modify the downloaded update in such a way that the digital signature is no longer valid. The evaluator will then attempt to install the modified update. The evaluator will ensure that the OS does not install the modified update.

High-Level Test Description
Using a package in the IPS manager, modify the binary to fail the signature verification activity. Attempt to install and show it fails. Using a package in the IPS manager, remove the signature from the package. Attempt to install and show it fails. Finally, attempt the install operation again with a known-good package and signature and show that the installation succeeds.
PASS

219 Test 2: The evaluator will ensure that the update has a digital signature belonging to the vendor. The evaluator will then attempt to install the update (or permit installation to continue). The evaluator will ensure that the OS successfully installs the update.

NOTE:	See previous test case.

3.4.6 FPT_TUD_EXT.2 Trusted Update for Application Software

FPT_TUD_EXT.2.1

3.4.6.1 Guidance Documentation

220 The evaluator will check for updates to application software using procedures described in the documentation.

Findings:	Guidance, best practices and appropriate commands regarding updating applications packaged through the IPS system are provided in the [SUPP] in section 2.4.

3.4.6.2 Tests

221 **This entire section modified by TD0463**

222 The evaluator will check for updates to application software using procedures described in the documentation and verify that the OS provides a list of available updates. Testing this capability may require installing and temporarily placing the system into a configuration in conflict with secure configuration guidance which specifies automatic update.

Findings:	Both OS and application software are delivered over identical repository technologies. There are no differences in functionality. Please refer to FPT_TUD_EXT.1.1.

223 The evaluator is also to ensure that the response to this query is authentic by using a digital signature scheme specified in FCS_COP.1(3). The digital signature verification may be performed as part of a network protocol described in FTP_ITC_EXT.1.) If the signature verification is not performed as part of a trusted channel, the evaluator shall send a query response with a bad signature and verify that the signature verification fails. The evaluator shall then send a query response with a good signature and verify that the signature verification is successful.

Findings:	Both OS and application software are delivered over identical repository technologies. There are no differences in functionality. Please refer to FPT_TUD_EXT.1.1.

FPT_TUD_EXT.2.2

3.4.6.3 TSS

224 The evaluator will initiate an update to an application. This may vary depending on the application, but it could be through the application vendor's website, a commercial app store, or another system. All origins supported by the OS must be indicated in the TSS and evaluated.

Findings:	Section 6.6.5 of the [ST] specifies that the TOE application software is distributed from IPS package repositories.

3.4.6.4 Tests

225 The evaluator will initiate an update to an application. This may vary depending on the application, but it could be through the application vendor's website, a commercial app store, or another system. All origins supported by the OS must be indicated in the TSS and evaluated. However, this only includes those mechanisms for which the OS is providing a trusted installation and update functionality. It does not include user or administrator-driven download and installation of arbitrary files.

226 Test 1: The evaluator will ensure that the update has a digital signature which chains to the OS vendor or another trusted root managed through the OS. The evaluator will modify the downloaded update in such a way that the digital signature is no longer valid. The evaluator will then attempt to install the modified update. The evaluator will ensure that the OS does not install the modified update.

Findings:	Both OS and application software are delivered over identical repository technologies. There are no differences in functionality. Please refer to FPT_TUD_EXT.1.2.

227 Test 2: The evaluator will ensure that the update has a digital signature belonging to the OS vendor or another trusted root managed through the OS. The evaluator will then attempt to install the update. The evaluator will ensure that the OS successfully installs the update.

Findings:	Both OS and application software are delivered over identical repository technologies. There are no differences in functionality. Please refer to FPT_TUD_EXT.1.2.

3.5 Security Audit (FAU)

3.5.1 FAU_GEN.1 Audit Data Generation (Refined)

FAU_GEN.1.1

3.5.1.1 Guidance Documentation

228 The evaluator will check the administrative guide and ensure that it lists all of the auditable events. The evaluator will check to make sure that every audit event type selected in the ST is included.

Findings:	All of the audit events are described in section 4.2 of the [SUPP]. The evaluator confirmed that all audit events in section 4.2 of [SUPP] are included from the [ST] and broken out into more specific entries where necessary for clarity.

3.5.1.2 Tests

229 The evaluator will test the OS's ability to correctly generate audit records by having the TOE generate audit records for the events listed in the ST. This should include all instance types of an event specified. When verifying the test results, the evaluator will ensure the audit records generated during testing match the format specified in the administrative guide, and that the fields in each audit record have the proper entries.

High-Level Test Description
Start-up and shut-down of the audit functions Stop the audit daemon and show there is an audit message generated. Start the audit daemon and show there is an audit message generated. *Authentication events are witnessed as part of FIA_UAU.5* *Use of privileged/special rights events (Successful and unsuccessful security, audit, and configuration changes) are witnessed as part of FMT_SMF_EXT.1 for appropriate functions.* *Privilege or role escalation events (Success/Failure) are partially witnessed in FIA_AFL.1* Execute a privileged command without the necessary privileges and show it is not successful and audited. Then provide the necessary privileges and show that the same command is successful when run within a privilege-aware context and audited. Execute a setuid binary and show that the privilege use is audited. *File and object events are partially witnessed in FDP_ACF_EXT.1.* For “modify permissions”: Log into the TOE as user1 and create a file with privileges 600. Show that user2 is unable to read it. Then use user1 to modify the permissions on the file to be 644 and show that user2 is able to read it. *User and Group management events (Successful and unsuccessful add, delete, modify, disable, enable, and credential change)* As an unprivileged user, attempt to create a new user, group and role and show they do not succeed and the result is audited. As a privileged user, attempt to create a new user, group and role and show they succeed and the result is audited. As an unprivileged user, attempt to modify a user, group and role and show that they do not succeed and the result is audited. As a privileged user, attempt to modify a user, group and role and show they succeed and the result is audited. As an unprivileged user, attempt to delete a user, group and role and show that they do not succeed and the result is audited. As a privileged user, attempt to delete a user, group and role and show they succeed and the result is audited. *Audit and log data access events (Success/Failure) is witnessed in FPT_ACF_EXT.1* *Cryptographic verification of software (Success/Failure) is witnessed in FPT_TUD_EXT.1 and FPT_TUD_EXT.2* *Attempted application invocation with arguments (Success/Failure e.g. due to software restriction policy) ,*

High-Level Test Description
Execute a binary which has execute bits and show it works. Execute a binary which is missing the execute bits and show it fails to run. Show that the audit log contains the invocation with arguments when the audit policy is configured appropriately. Repeat this using the kernel API. *System reboot, restart, and shutdown events (Success/Failure) ,* Execute all methods of restarting and shutting down the machine with both privileged and unprivileged users. Show the successful and unsuccessful attempts are audited. *Kernel module load/unload (success/failure)* As an unprivileged user, attempt to load a module and unload a module. Show the attempts fail and are audited. As a privileged user, attempt to load and unload a module. Show the attempts succeed and are audited. As a privileged user, attempt to load an unsigned module. Show the attempt fails and is audited. *Administrator or root-level access events (Success/Failure) are witnessed as authentication events and privilege escalation events above.* For each event, verify the information contains, minimally, the date, subject and outcome.
PASS

FAU_GEN.1.2

3.5.1.3 Guidance Documentation

230 The evaluator will check the administrative guide and ensure that it provides a format for audit records. Each audit record format type must be covered, along with a brief description of each field. The evaluator will ensure that the fields contains the information required.

Findings:	Section 4.1 of the [SUPP] provides a reference to the broader “Managing Auditing in Oracle® Solaris 11.4” reference which includes a format of the audit records in chapter 7: “Auditing Reference“, subsection “Audit Record Structure”. The structure of the audit records is given at a high level. Specifically of note, the audit records are stored in a binary format and denoted by “tokens”. Each token has a specific informational context described therein (and further expanded upon in audit.log(5)). The binary format is interpreted using a combination of reduction (via auditreduce to filter) and praudit (to interpret the binary format). The binary format is converted to human-readable format according to the tokens contained in each audit record. The reference in “Managing Auditing in Oracle® Solaris 11.4” describes the use of the tool “auditrecord” which can describe each audit record format individually with a description of each field specific to that record.

The TOE is required to record the date and time of the event, the type of event, the subject identity (if applicable) and the outcome (success or failure). It is also required to contain any other specific information noted by the ST author. For this TOE, no other information is mandated.

The date and time of the event is included in every event within the “header” (or “expanded header”) token, which is included in every record. audit.log(5) shows that the header/expanded header token has the “seconds of time” and “nanoseconds of time” which are based on a Unix epoch. praudit interprets the information and presents the time information as a human-readable timestamp (2020-10-28 23:07:33.644-07:00).

The type of event is produced by praudit as a human-readable string based on the format of audit_event(5). The auditrecord(8) tool can help an administrator understand the audit formats specific to each audit type.

The subject identity is part of any audit event which operates with a subject. For example, for login and logout audit events, we can see that auditrecord(8) provides the format definition as:

```
terminal login
  program     /usr/sbin/login      See login(1)
              /usr/sbin/gdm        See gdm(8)
  event ID    6152                 AUE_login
  class       lo                   (0x0000000000001000)
      header
      subject
      return


login: logout
  program     various              See login(1)
  event ID    6153                 AUE_logout
  class       lo                   (0x0000000000001000)
      header
      subject
      return
```

The subject is intrinsically tied to the type of event. For non-attributable events, the subject may be missing when it is not available, such as when the audit daemon starts up (the system starts it, rather than a subject operating on behalf of a user).

Finally, the audit format records have a token called “return” which indicates the return status of the action (success or failure).

3.5.1.4 Tests

231 The evaluator shall test the OS's ability to correctly generate audit records by having the TOE generate audit records for the events listed in the ST. The evaluator will ensure the audit records generated during testing match the format specified in the administrative guide, and that the fields in each audit record provide the required information.

Note	Please see previous test case.

3.6 Identification and Authentication (FIA)

3.6.1 FIA_AFL.1 Authentication Failure Management

FIA_AFL.1.1

3.6.1.1 Tests

232 The evaluator will set an administrator-configurable threshold for failed attempts, or note the ST-specified assignment. The evaluator will then (per selection) repeatedly attempt to authenticate with an incorrect password, PIN, or certificate until the number of attempts reaches the threshold. Note that the authentication attempts and lockouts must also be logged as specified in FAU_GEN.1.

High-Level Test Description
For each defined use of the authentication mechanism (see FIA_UAU.5) that uses passwords, define the lockout thresholds and then lock the account. Show that the account is locked, then perform the required actions to restore the account and show it is restored. This test case is performed for both manual unlocking and time-based unlocking.
PASS

FIA_AFL.1.2

3.6.1.2 Guidance Documentation

233 Application Note: The action to be taken shall be populated in the assignment of the ST and defined in the administrator guidance.

Findings:	Section 3.3.4 of the [SUPP] provides information about what happens when the authentication failure limit is reached (account is locked). The same section indicates that if the login/auto_unlock_time is set, then the account will be unlocked automatically after the allotted time. However, in the event an account needs to be unlocked manually, section 3.3.4.3 of the [SUPP] indicates that user accounts can be unlocked using the *passwd -u* CLI command.

3.6.1.3 Tests

234 Test 1: The evaluator will attempt to authenticate repeatedly to the system with a known bad password. Once the defined number of failed authentication attempts has been reached the evaluator will ensure that the account that was being used for testing has had the actions detailed in the assignment list above applied to it. The evaluator will ensure that an event has been logged to the security event log detailing that the account has had these actions applied.

NOTE:	See previous test case.

235 Test 2: The evaluator will attempt to authenticate repeatedly to the system with a known bad certificate. Once the defined number of failed authentication attempts has been reached the evaluator will ensure that the account that was being used for testing has had the actions detailed in the assignment list above applied to it. The evaluator will ensure that an event has been logged to the security event log detailing that the account has had these actions applied.

NOTE:	Not applicable. The TOE does not claim certificate-based authentication mechanisms.

236 Test 3: The evaluator will attempt to authenticate repeatedly to the system using both a bad password and a bad certificate. Once the defined number of failed authentication attempts has been reached the evaluator will ensure that the account that was being used for testing has had the actions detailed in the assignment list above applied to it. The evaluator will ensure that an event has been logged to the security event log detailing that the account has had these actions applied.

NOTE:	Not applicable. The TOE does not claim certificate-based authentication mechanisms.

3.6.2 FIA_UAU.5 Multiple Authentication Mechanisms (Refined)

FIA_UAU.5.1

3.6.2.1 Tests

237 If user name and password authentication is selected, the evaluator will configure the OS with a known user name and password and conduct the following tests:

- Test 1: The evaluator will attempt to authenticate to the OS using the known user name and password. The evaluator will ensure that the authentication attempt is successful.

NOTE:	Please refer to FIA_AFL.1 test cases which show positive password-based authentication.

- Test 2: The evaluator will attempt to authenticate to the OS using the known user name but an incorrect password. The evaluator will ensure that the authentication attempt is unsuccessful.

NOTE:	Please refer to FIA_AFL.1 test cases which show negative password-based authentication.

238 If user name and PIN that releases an asymmetric key is selected, the evaluator will examine the TSS for guidance on supported protected storage and will then configure the TOE or OE to establish a PIN which enables release of the asymmetric key from the protected storage (such as a TPM, a hardware token, or isolated execution environment) with which the OS can interface. The evaluator will then conduct the following tests:

- Test 1: The evaluator will attempt to authenticate to the OS using the known user name and PIN. The evaluator will ensure that the authentication attempt is successful.
- Test 2: The evaluator will attempt to authenticate to the OS using the known user name but an incorrect PIN. The evaluator will ensure that the authentication attempt is unsuccessful.

Findings:	The TOE does not claim this mechanism.

239 If X.509 certificate authentication is selected, the evaluator will generate an X.509v3 certificate for a user with the Client Authentication Enhanced Key Usage field set. The evaluator will provision the OS for authentication with the X.509v3 certificate. The evaluator will ensure that the certificates are validated by the OS as per FIA_X509_EXT.1.1 and then conduct the following tests:

- Test 1: The evaluator will attempt to authenticate to the OS using the X.509v3 certificate. The evaluator will ensure that the authentication attempt is successful.
- Test 2: The evaluator will generate a second certificate identical to the first except for the public key and any values derived from the public key. The evaluator will attempt to authenticate to the OS with this certificate. The evaluator will ensure that the authentication attempt is unsuccessful.

Findings:	The TOE does not claim this mechanism.

FIA_UAU.5.2

3.6.2.2 TSS

240 The evaluator will ensure that the TSS describes each mechanism provided to support user authentication and the rules describing how the authentication mechanism(s) provide authentication.

Findings:	Section 6.4.2 of the [ST] specifies that the TOE uses password based authentication and compares the hash of the entered value against the hash in the local database to ensure that they are correct. Further that the TOE uses public key authentication which requires ownership of the public key in the authorized key file.

3.6.2.3 Guidance Documentation

241 The evaluator will verify that configuration guidance for each authentication mechanism is addressed in the AGD guidance.

Findings:	Each authentication mechanism is described in section 3.3.2 of the [SUPP]. Passwords are stored the /etc/shadow credential store. SSH public keys are placed in the user's designated authorized_keys file as per the man page in sshd(8).

3.6.2.4 Tests

242 Test 1: For each authentication mechanism selected, the evaluator will enable that mechanism and verify that it can be used to authenticate the user at the specified authentication factor interfaces.

NOTE:	Refer to test cases in FIA_AFL.1, FCS_SSHS_EXT.1 and FTP_TRP.1 which show behaviour for both password-based and public key-based authentication for all interfaces.

243 Test 2: For each authentication mechanism rule, the evaluator will ensure that the authentication mechanism(s) behave as documented in the TSS.

NOTE:	Refer to test cases in FIA_AFL.1 and FCS_SSHS_EXT.1 and FTP_TRP.1 which show behaviour for both password-based and public key-based authentication for all interfaces under all designated rules.

3.6.3 FIA_X509_EXT.1 X.509 Certificate Validation

FIA_X509_EXT.1.1

3.6.3.1 TSS

244 **This entire section modified by TD0525**

245 The evaluator will ensure the TSS describes where the check of validity of the certificates takes place. The evaluator ensures the TSS also provides a description of the certificate path validation algorithm.

Findings:	Section 6.4.3 of the [ST] specifies that certification validation takes place during TLS client inspection of the server X.509 certificate. Section 6.4.3 of the [ST] also describes the certificate path validation.

3.6.3.2 Tests

246 **This entire section modified by TD0525**

247 The tests described must be performed in conjunction with the other certificate services evaluation activities, including the functions in FIA_X509_EXT.2.1. The tests for the extendedKeyUsage rules are performed in conjunction with the uses that require those rules. The evaluator will create a chain of at least four certificates: the node certificate to be tested, two Intermediate CAs, and the self-signed Root CA.

248 Test 1: The evaluator shall demonstrate that validating a certificate without a valid certification path results in the function failing, for each of the following reasons, in turn:

- by establishing a certificate path in which one of the issuing certificates is not a CA certificate,
- by omitting the basicConstraints field in one of the issuing certificates,
- by setting the basicConstraints field in an issuing certificate to have CA=False,
- by omitting the CA signing bit of the key usage field in an issuing certificate, and
- by setting the path length field of a valid CA field to a value strictly less than the certificate path.

249 The evaluator shall then establish a valid certificate path consisting of valid CA certificates, and demonstrate that the function succeeds. The evaluator shall then remove trust in one of the CA certificates, and show that the function fails.

High-Level Test Description
Show that when the chain is properly constructed, the TOE TLS client can connect. Construct the following certificates: - A leaf that has been signed by another leaf certificate; - An issuing certificate which has the CAsign flag omitted in the keyUsage extension; - An intermediate certificate which has the pathlength set to 0 such that its child is not a leaf For each of these certificates, show that the TOE fails to connect. *Note that testing involving basicConstraint checks are performed in FIA_X509_EXT.1.2.*
PASS

250 Test 2: The evaluator will demonstrate that validating an expired certificate results in the function failing.

High-Level Test Description
Using a TOE TLS client, connect to a TLS server which will return back an expired certificate and show the connection fails. Using an expired CA in the trust store, show that the TOE TLS client fails to connect to the TLS server.
PASS

251 Test 3: The evaluator will test that the OS can properly handle revoked certificates - conditional on whether CRL, OCSP, OCSP stapling, or OCSP multi-stapling is selected; if multiple methods are selected, then a test shall be performed for each

method. The evaluator will test revocation of the node certificate and revocation of the intermediate CA certificate (i.e. the intermediate CA certificate should be revoked by the root CA). If OCSP stapling per RFC 6066 is the only supported revocation method, testing revocation of the intermediate CA certificate is omitted. The evaluator will ensure that a valid certificate is used, and that the validation function succeeds. The evaluator then attempts the test with a certificate that has been revoked (for each method chosen in the selection) to ensure when the certificate is no longer valid that the validation function fails.

High-Level Test Description
Using a TOE TLS client, make a connection to a TLS server in the TOE environment using CRLs. Show that when a certificate is flagged as revoked in the CRL, the connection fails. When the certificate is not flagged as revoked in the CRL, the connection succeeds. Using a TOE TLS client, make a connection to a TLS server in the TOE environment using an OCSP responder. Show that when a certificate is not flagged as valid in the OCSP response, the connection fails. When the certificate is flagged as valid in the OCSP response, the connection succeeds.
PASS

252 Test 4: If any OCSP option is selected, the evaluator shall configure the OCSP server or use a man-in-the-middle tool to present a certificate that does not have the OCSP signing purpose and verify that validation of the OCSP response fails. If CRL is selected, the evaluator shall configure the CA to sign a CRL with a certificate that does not have the cRLsign key usage bit set and verify that validation of the CRL fails.

High-Level Test Description
Using a TLS client, connect to the TLS server and verify that the TLS client will fail to validate the OCSP response or CRL, respectively, when the OCSP or CRL is signed by a CA which does not have the proper policy flag extension set.
PASS

253 Test 5: The evaluator shall modify any byte in the first eight bytes of the certificate and demonstrate that the certificate fails to validate. (The certificate will fail to parse correctly.)

High-Level Test Description
Force the TOE to connect to a Lightship test server which will send back a properly mangled X.509 certificate in which the ASN.1 header bytes in the first 8 bytes are modified.
PASS

254 Test 6: The evaluator shall modify any byte in the last byte of the certificate and demonstrate that the certificate fails to validate. (The signature on the certificate will not validate.)

High-Level Test Description
Force the TOE to connect to a Lightship test server which will send back an X.509 certificate in which the last byte of the certificate (the signature) is modified.
PASS

255 Test 7: The evaluator shall modify any byte in the public key of the certificate and demonstrate that the certificate fails to validate. (The signature of the certificate will not validate.)

High-Level Test Description
Force the TOE to connect to a Lightship test server which will send back an X.509 certificate in which the public key of the certificate is modified.
PASS

256 Test 8a: (Conditional on support for EC certificates as indicated in FCS_COP.1(3)). The evaluator shall establish a valid, trusted certificate chain consisting of an EC leaf certificate, an EC Intermediate CA certificate not designated as a trust anchor, and an EC certificate designated as a trusted anchor, where the elliptic curve parameters are specified as a named curve. The evaluator shall confirm that the TOE validates the certificate chain.

257 Test 8b: (Conditional on support for EC certificates as indicated in FCS_COP.1(3)). The evaluator shall replace the intermediate certificate in the certificate chain for Test 8a with a modified certificate, where the modified intermediate CA has a public key information field where the EC parameters uses an explicit format version of the Elliptic Curve parameters in the public key information field of the intermediate CA certificate from Test 8a, and the modified Intermediate CA certificate is signed by the trusted EC root CA, but having no other changes. The evaluator shall confirm the TOE treats the certificate as invalid.

Note:	While the TOE supports ECDSA, the TLS channel which can make use of ECDSA X.509 certificates does not support any ciphersuite which uses ECDSA X.509 certificates for authentication. Therefore, this test case is not required.

FIA_X509_EXT.1.2

3.6.3.3 Tests

258 The tests described must be performed in conjunction with the other certificate services evaluation activities, including the functions in FIA_X509_EXT.2.1. The evaluator will create a chain of at least four certificates: the node certificate to be tested, two Intermediate CAs, and the self-signed Root CA.

259 Test 1: The evaluator will construct a certificate path, such that the certificate of the CA issuing the OS's certificate does not contain the basicConstraints extension. The validation of the certificate path fails.

High-Level Test Description
Using a TLS client, connect to a TLS server such that the issuing Intermediate CA fails to validate when the trust anchor is missing the basicConstraint extension.
PASS

260 Test 2: The evaluator will construct a certificate path, such that the certificate of the CA issuing the OS's certificate has the CA flag in the basicConstraints extension not set. The validation of the certificate path fails.

High-Level Test Description
Using a TLS client, connect to a TLS server such that the issuing Intermediate CA fails to validate when the trust anchor has a basicConstraint extension in which the CA flag is set to FALSE.
PASS

261 Test 3: The evaluator will construct a certificate path, such that the certificate of the CA issuing the OS's certificate has the CA flag in the basicConstraints extension set to TRUE. The validation of the certificate path succeeds.

Note:	This test succeeded by virtue of the other X.509 tests completing successfully. Use of a CA=True flag in the basicConstraints is a default property of the generated certificates.

3.6.4 FIA_X509_EXT.2 X.509 Certificate Authentication

3.6.4.1 Tests

262 The evaluator will acquire or develop an application that uses the OS TLS mechanism with an X.509v3 certificate. The evaluator will then run the application and ensure that the provided certificate is used to authenticate the connection.

High-Level Test Description
Using a TLS client with a client X.509 certificate, connect to the TLS server in the environment and show that the X.509 certificate from the TLS client is used to authenticate to the non-TOE server.
PASS

263 The evaluator will repeat the activity for any other selections listed.

Findings:	No other selections are listed.

3.7 Trusted path/channels (FTP)

3.7.1 FTP_ITC_EXT.1 Trusted channel communication

3.7.1.1 Tests

264 The evaluator will configure the OS to communicate with another trusted IT product as identified in the second selection. The evaluator will monitor network traffic while the OS performs communication with each of the servers identified in the second

selection. The evaluator will ensure that for each session a trusted channel was established in conformance with the protocols identified in the first selection.

Note	This test case is performed throughout FCS_TLSC_EXT.1 and FCS_SSHS_EXT.1.

3.7.2 FTP_TRP.1 Trusted Path

3.7.2.1 TSS

265 The evaluator will examine the TSS to determine that the methods of remote OS administration are indicated, along with how those communications are protected. The evaluator will also confirm that all protocols listed in the TSS in support of OS administration are consistent with those specified in the requirement, and are included in the requirements in the ST.

Findings:	Section 6.2.7 of the [ST] specifies that the TOE uses SSH to secure remote administrator communications. This is consistent with the selections made in FTP_ITC_EXT.1 as per the application note for FTP_TRP.1. The [SSHEP] SFRs have been included in the [ST].

3.7.2.2 Guidance Documentation

266 The evaluator will confirm that the operational guidance contains instructions for establishing the remote administrative sessions for each supported method.

Findings:	This information is provided in section 3.3.1 of the [SUPP]. SSH is the only applicable remote OS administrative trusted path. Establishing a session requires the use of an SSH client to interface with the TOE's SSH server. The means for using credential types are described in section 3.3.2 of [SUPP].

3.7.2.3 Tests

267 The evaluator will also perform the following tests:

268 Test 1: The evaluator will ensure that communications using each remote administration method is tested during the course of the evaluation, setting up the connections as described in the operational guidance and ensuring that communication is successful.

Note	All trusted paths are configured as per the evaluated configuration. They are constantly tested throughout the evaluation.

269 Test 2: For each method of remote administration supported, the evaluator will follow the operational guidance to ensure that there is no available interface that can be used by a remote user to establish a remote administrative sessions without invoking the trusted path.

High-Level Test Description
Perform a port scan of the device and determine if there are any remote administrative interfaces available outside of the SSH CLI interface.
PASS

270 Test 3: The evaluator will ensure, for each method of remote administration, the channel data is not sent in plaintext.

High-Level Test Description
Using a packet sniffer, show that the channel data is not being sent in plaintext for each of the TSFI.
PASS

271 Test 4: The evaluator will ensure, for each method of remote administration, modification of the channel data is detected by the OS.

High-Level Test Description
Using a man-in-the-middle attack mechanism, modify traffic destined to the TOE's remote administration interfaces and show that the modifications are detected.
PASS

4 Evaluation Activities for SARs

4.1 Class ASE: Security Target

272 The following ASE components as defined in [CEM] are required:

- Conformance claims (ASE_CCL.1)
- Extended components definition (ASE_ECD.1)
- ST introduction (ASE_INT.1)
- Security objectives (ASE_OBJ.2)
- Derived security requirements (ASE_REQ.2)
- Security Problem Definition (ASE_SPD.1)
- TOE summary specification (ASE_TSS.1)

273 The requirements for exact conformance of the Security Target are described in Section 2 Conformance Claims.

4.2 Class ADV: Development

274 The information about the OS is contained in the guidance documentation available to the end user as well as the TSS portion of the ST. The OS developer must concur with the description of the product that is contained in the TSS as it relates to the functional requirements. The evaluation activities contained in Section 5.1 Security Functional Requirements should provide the ST authors with sufficient information to determine the appropriate content for the TSS section.

4.2.1 ADV_FSP.1 Basic Functional Specification

275 The functional specification describes the TSFIs. It is not necessary to have a formal or complete specification of these interfaces. Additionally, because OSes conforming to this PP will necessarily have interfaces to the Operational Environment that are not directly invokable by OS users, there is little point specifying that such interfaces be described in and of themselves since only indirect testing of such interfaces may be possible. For this PP, the activities for this family should focus on understanding the interfaces presented in the TSS in response to the functional requirements and the interfaces presented in the AGD documentation. No additional "functional specification" documentation is necessary to satisfy the evaluation activities specified. The interfaces that need to be evaluated are characterized through the information needed to perform the assurance activities listed, rather than as an independent, abstract list.

4.2.1.1 Activities

276 There are no specific evaluation activities associated with these SARs, except ensuring the information is provided. The functional specification documentation is provided to support the evaluation activities described in Section 5.1 Security Functional Requirements, and other activities described for AGD, ATE, and AVA SARs. The requirements on the content of the functional specification information is implicitly assessed by virtue of the other evaluation activities being performed; if the

evaluator is unable to perform an activity because there is insufficient interface information, then an adequate functional specification has not been provided.

Findings:	For all test cases, the evaluator was able to exercise the appropriate interface based on information provided in the Oracle Solaris 11.4 documentation library.

4.3 Class AGD: Guidance Documentation

277 The guidance documents will be provided with the ST. Guidance must include a description of how the IT personnel verifies that the Operational Environment can fulfill its role for the security functionality. The documentation should be in an informal style and readable by the IT personnel. Guidance must be provided for every operational environment that the product supports as claimed in the ST. This guidance includes instructions to successfully install the TSF in that environment; and Instructions to manage the security of the TSF as a product and as a component of the larger operational environment. Guidance pertaining to particular security functionality is also provided; requirements on such guidance are contained in the Evaluation Activities specified with each requirement.

4.3.1 AGD_OPE.1 Operational User Guidance

4.3.1.1 Activities

278 Some of the contents of the operational guidance are verified by the evaluation activities in Section 5.1 Security Functional Requirements and evaluation of the OS according to the [CEM]. The following additional information is also required. If cryptographic functions are provided by the OS, the operational guidance shall contain instructions for configuring the cryptographic engine associated with the evaluated configuration of the OS.

Findings:	Section 3.4 of the [SUPP] provides information on configuring the FIPS 140-2 support within the TOE. This involves enabling the correct instance of OpenSSL as well as enabling explicit FIPS 140-2 support within the OS itself via the cryptoadm tool.

279 It shall provide a warning to the administrator that use of other cryptographic engines was not evaluated nor tested during the CC evaluation of the OS.

Findings:	Section 3.4 of the [SUPP] provides this warning.

280 The documentation must describe the process for verifying updates to the OS by verifying a digital signature – this may be done by the OS or the underlying platform. The evaluator will verify that this process includes the following steps: Instructions for obtaining the update itself. This should include instructions for making the update accessible to the OS (e.g., placement in a specific directory). Instructions for initiating the update process, as well as discerning whether the process was successful or unsuccessful. This includes generation of the hash/digital signature.

Findings:	Section 2 of the [SUPP] provides the necessary information needed to help an administrator verify updates to the OS via digital signatures. The source of updates is from the IPS package depot available locally or remotely (and ultimately sourced from the Oracle IPS support repository with a support contract). The update process can be initiated via “pkg update” and the use of the verbose flag (-v) enables the end-user to troubleshoot any issues if they arise.

281 The OS will likely contain security functionality that does not fall in the scope of evaluation under this PP. The operational guidance shall make it clear to an administrator which security functionality is covered by the evaluation activities.

Findings:	Section 1.3.3 of the [SUPP] provides a summary of the evaluated functionality.

4.3.2 AGD_PRE.1 Preparative Procedures

4.3.2.1 Activities

282 As indicated in the introduction above, there are significant expectations with respect to the documentation—especially when configuring the operational environment to support OS functional requirements. The evaluator shall check to ensure that the guidance provided for the OS adequately addresses all platforms claimed for the OS in the ST.

Findings:	The [ST] describes two platforms: T8 and X8. The T8 is a SPARC platform and the X8 is an Intel x86-64 platform. The SPARC platform has hardware differences over the X86 which appear to be adequately addressed in the supplied user documentation. Under section 1.5 of the [SUPP], there is a pointer to the main Solaris 11.4 Information Library. Within this library, there is a section called “Installing and Booting Oracle Solaris” which describes the differences between the two platforms. In another section titled “Securing the Oracle Solaris Operating System”, there is a comprehensive treatment of verified boot and how it differs between the two platforms.

4.4 Class ALC: Life-cycle Support

283 At the assurance level provided for OSes conformant to this PP, life-cycle support is limited to end-user-visible aspects of the life-cycle, rather than an examination of the OS vendor's development and configuration management process. This is not meant to diminish the critical role that a developer's practices play in contributing to the overall trustworthiness of a product; rather, it is a reflection on the information to be made available for evaluation at this assurance level.

4.4.1 ALC_CMC.1 Labeling of the TOE

4.4.1.1 Activities

284 The evaluator will check the ST to ensure that it contains an identifier (such as a product name/version number) that specifically identifies the version that meets the requirements of the ST. Further, the evaluator will check the AGD guidance and OS samples received for testing to ensure that the version number is consistent with that in the ST. If the vendor maintains a web site advertising the OS, the evaluator will examine the information on the web site to ensure that the information in the ST is sufficient to distinguish the product.

Findings:	The [ST] contains the product name and reference in section 1.2. The [SUPP] provides the same information in section 1.3.2. Finally, the TOE was verified using the instructions provided in section 2.2 of the [SUPP] and was found to be consistent with the claimed version.

	Oracle maintains a website for advertising Solaris and the information in the ST is sufficient to distinguish the product. Specifically, the version on the web site is 11.4. The SRUs and IDRs are only available to those with support contracts and they would be able to acquire the TOE through those channels. The documentation provided by Oracle for the TOE is clearly designated for Solaris 11.4.

4.4.2 ALC_CMS.1 TOE CM Coverage

285 Given the scope of the OS and its associated evaluation evidence requirements, this component's evaluation activities are covered by the evaluation activities listed for ALC_CMC.1.

4.4.2.1 Activities

286 The "evaluation evidence required by the SARs" in this PP is limited to the information in the ST coupled with the guidance provided to administrators and users under the AGD requirements. By ensuring that the OS is specifically identified and that this identification is consistent in the ST and in the AGD guidance (as done in the evaluation activity for ALC_CMC.1), the evaluator implicitly confirms the information required by this component. Life-cycle support is targeted aspects of the developer's life-cycle and instructions to providers of applications for the developer's devices, rather than an in-depth examination of the TSF manufacturer's development and configuration management process. This is not meant to diminish the critical role that a developer's practices play in contributing to the overall trustworthiness of a product; rather, it's a reflection on the information to be made available for evaluation.

287 The evaluator will ensure that the developer has identified (in guidance documentation for application developers concerning the targeted platform) one or more development environments appropriate for use in developing applications for the developer's platform. For each of these development environments, the developer shall provide information on how to configure the environment to ensure that buffer overflow protection mechanisms in the environment(s) are invoked (e.g., compiler and linker flags).

Findings:	Section 1.5 of the [SUPP] provides a pointer into the larger document library for Solaris 11.4. Within this library, there are significant resources available to developers. Specifically, in the section “Developing Applications For Use With Oracle Solaris”, there is information about several application development environments. For application development environments which produce binary machine code, the linker ld(1) provides link-time flags to explicitly enable aslr, nxstack and nxheap security extensions (-z sx=aslr -z sx=nxstack -z sx=nxheap). Note, however, that these security extensions are available by default in the TOE even if these flags are not provided (as per sxadm(8) which can be found by reviewing [INFO] under “man pages section 8: System Administration Commands”).

288 The evaluator will ensure that this documentation also includes an indication of whether such protections are on by default, or have to be specifically enabled.

Findings:	The nxstack, nxheap and aslr link-time security extensions are available by default in the TOE even if these flags are not provided (as per the information provided in sxadm(8) which can be found by reviewing [INFO] under “man pages section 8: System Administration Commands”).

289 The evaluator will ensure that the TSF is uniquely identified (with respect to other products from the TSF vendor), and that documentation provided by the developer in association with the requirements in the ST is associated with the TSF using this unique identification.

Findings:	Please refer to ALC_CMC.1.

4.4.3 ALC_TSU_EXT.1 Timely Security Updates

290 This component requires the OS developer, in conjunction with any other necessary parties, to provide information as to how the end-user devices are updated to address security issues in a timely manner. The documentation describes the process of providing updates to the public from the time a security flaw is reported/discovered, to the time an update is released. This description includes the parties involved (e.g., the developer, carriers(s)) and the steps that are performed (e.g., developer testing, carrier testing), including worst case time periods, before an update is made available to the public.

4.4.3.1 Activities

291 The evaluator will verify that the TSS contains a description of the timely security update process used by the developer to create and deploy security updates. The evaluator will verify that this description addresses the entire application. The evaluator will also verify that, in addition to the OS developer's process, any third-party processes are also addressed in the description. The evaluator will also verify that each mechanism for deployment of security updates is described.

Findings:	Section 5.4.2 of the [ST] provides links to the developer's "timely security update methodology". The evaluator reviewed previous updates from the "security alerts" website found in section 5.4.2 of the [ST] and verified that third party applications are included in the updates. The TOE only provides a single mechanism for deployment of updates.

292 The evaluator will verify that, for each deployment mechanism described for the update process, the TSS lists a time between public disclosure of a vulnerability and public availability of the security update to the OS patching this vulnerability, to include any third-party or carrier delays in deployment. The evaluator will verify that this time is expressed in a number or range of days.

Findings:	The developer's "timely security update methodology" website described in section 5.4.2 of the [ST] notes that it is Oracle's policy to announce security fixes as much as possible only when the fixes are available for all affected and supported product version and platform combinations. The same website further notes that "Minor delays in patch availability for up to two weeks from the announcement date generally due to technical issues during the production or testing of the patch".

293 The evaluator will verify that this description includes the publicly available mechanisms (including either an email address or website) for reporting security issues related to the OS. The evaluator shall verify that the description of this mechanism includes a method for protecting the report either using a public key for encrypting email or a trusted channel for a website.

Findings:	Reporting is also described in the “security vulnerability reporting procedures” website described in section 5.4.2 of the [ST]. The suggested method is emailing the “secalert_us@oracle.com”. The PGP key is published with the email address.

4.5 Class ATE: Tests

294 Testing is specified for functional aspects of the system as well as aspects that take advantage of design or implementation weaknesses. The former is done through the ATE_IND family, while the latter is through the AVA_VAN family. At the assurance level specified in this PP, testing is based on advertised functionality and interfaces with dependency on the availability of design information. One of the primary outputs of the evaluation process is the test report as specified in the following requirements.

4.5.1 ATE_IND.1 Independent Testing

295 Testing is performed to confirm the functionality described in the TSS as well as the administrative (including configuration and operational) documentation provided. The focus of the testing is to confirm that the requirements specified in Section 5.1 Security Functional Requirements being met, although some additional testing is specified for SARs in Section 5.2 Security Assurance Requirements. The evaluation activities identify the additional testing activities associated with these components. The evaluator produces a test report documenting the plan for and results of testing, as well as coverage arguments focused on the platform/OS combinations that are claiming conformance to this PP. Given the scope of the OS and its associated evaluation evidence requirements, this component's evaluation activities are covered by the evaluation activities listed for ALC_CMC.1.

4.5.1.1 Activities

296 The evaluator will prepare a test plan and report documenting the testing aspects of the system, including any application crashes during testing. The evaluator shall determine the root cause of any application crashes and include that information in the report. The test plan covers all of the testing actions contained in the [CEM] and the body of this PP's evaluation activities.

297 While it is not necessary to have one test case per test listed in an evaluation activity, the evaluator must document in the test plan that each applicable testing requirement in the ST is covered. The test plan identifies the platforms to be tested, and for those platforms not included in the test plan but included in the ST, the test plan provides a justification for not testing the platforms. This justification must address the differences between the tested platforms and the untested platforms, and make an argument that the differences do not affect the testing to be performed. It is not sufficient to merely assert that the differences have no affect; rationale must be provided. If all platforms claimed in the ST are tested, then no rationale is necessary. The test plan describes the composition of each platform to be tested, and any setup that is necessary beyond what is contained in the AGD documentation. It should be noted that the evaluator is expected to follow the AGD documentation for installation and setup of each platform either as part of a test or as a standard pre-test condition. This may include special test drivers or tools. For each driver or tool, an argument (not just an assertion) should be provided that the driver or tool will not adversely affect the performance of the functionality by the OS and its platform.

298 This also includes the configuration of the cryptographic engine to be used. The cryptographic algorithms implemented by this engine are those specified by this PP and used by the cryptographic protocols being evaluated (IPsec, TLS). The test plan identifies high-level test objectives as well as the test procedures to be followed to achieve those objectives. These procedures include expected results.

299 The test report (which could just be an annotated version of the test plan) details the activities that took place when the test procedures were executed, and includes the actual results of the tests. This shall be a cumulative account, so if there was a test run that resulted in a failure; a fix installed; and then a successful re-run of the test, the report would show a "fail" and "pass" result (and the supporting details), and not just the "pass" result.

Findings:	A test report was produced in accordance with the requirements. In cases where faults were discovered, the “findings” include an assessment of the original attempt, the purported fix and a re-assessment of the fix to ensure it corrected the original deficiency.

4.6 Class AVA: Vulnerability Assessment

300 For the first generation of this protection profile, the evaluation lab is expected to survey open sources to discover what vulnerabilities have been discovered in these types of products. In most cases, these vulnerabilities will require sophistication beyond that of a basic attacker. Until penetration tools are created and uniformly distributed to the evaluation labs, the evaluator will not be expected to test for these vulnerabilities in the OS. The labs will be expected to comment on the likelihood of these vulnerabilities given the documentation provided by the vendor. This information will be used in the development of penetration testing tools and for the development of future protection profiles.

4.6.1 AVA_VAN.1 Vulnerability Survey

4.6.1.1 Activities

301 The evaluator will generate a report to document their findings with respect to this requirement. This report could physically be part of the overall test report mentioned in ATE_IND, or a separate document. The evaluator performs a search of public information to find vulnerabilities that have been found in similar applications with a particular focus on network protocols the application uses and document formats it parses. The evaluator documents the sources consulted and the vulnerabilities found in the report.

302 For each vulnerability found, the evaluator either provides a rationale with respect to its non-applicability, or the evaluator formulates a test (using the guidelines provided in ATE_IND) to confirm the vulnerability, if suitable. Suitability is determined by assessing the attack vector needed to take advantage of the vulnerability. If exploiting the vulnerability requires expert skills and an electron microscope, for instance, then a test would not be suitable and an appropriate justification would be formulated.

Findings:	A vulnerability report and penetration test plan were produced in accordance with the requirements. For each vulnerability found a rationale was offered to ensure it is clear the vulnerability is mitigated or not applicable.

5 Evaluation Activities for Optional Requirements

303 No optional requirements have been selected.

6 Evaluation Activities for Selection-Based Requirements

6.1 Cryptographic Support (FCS)

6.1.1 FCS_COP.1/SSH FCS_COP.1/SSH Cryptographic Operation - Encryption/Decryption (Refined)

6.1.1.1 TSS

304 **TD0240 -** The evaluator shell review the TSF of the base PP to verify consistency with the functionality that was claimed by the base PP to ensure that applicable dependencies are met.

Findings:	Section 6.2.7 of the [ST] was reviewed and found consistent with the selections in the associated SFR.

305 **TD0240 -** If perform encryption/decryption services is chosen, the evaluator shall verify that the TSS describes the counter mechanism including rationale that the counter values provided are unique.

Findings:	Section 6.2.3 of the [ST] specifies that the TOE implements a counter based on the Standard Incrementing Function as defined in Appendix B.1 of NIST SP800-38a. The counter values are unique as they start from a random Initialization Vector (IV).

6.1.1.2 Tests

306 **This whole section modified by TD0240**

AES-CTR Tests:

307 Test 1: Known Answer Tests (KATs)

308 There are four Known Answer Tests (KATs) described below. For all KATs, the plaintext, IV, and ciphertext values shall be 128-bit blocks. The results from each test may either be obtained by the validator directly or by supplying the inputs to the implementer and receiving the results in response. To determine correctness, the evaluator shall compare the resulting values to those obtained by submitting the same inputs to a known good implementation.

309 To test the encrypt functionality, the evaluator shall supply a set of 10 plaintext values and obtain the ciphertext value that results from encryption of the given plaintext using a key value of all zeros and an IV of all zeros. Five plaintext values shall be encrypted with a 128-bit all zeros key, and the other five shall be encrypted with a 256-bit all zeros key. To test the decrypt functionality, the evaluator shall perform the same test as for encrypt, using 10 ciphertext values as input.

310 To test the encrypt functionality, the evaluator shall supply a set of 10 key values and obtain the ciphertext value that results from encryption of an all zeros plaintext using the given key value and an IV of all zeros. Five of the key values shall be 128-bit keys, and the other five shall be 256-bit keys. To test the decrypt functionality, the evaluator shall perform the same test as for encrypt, using an all zero ciphertext value as input.

311 To test the encrypt functionality, the evaluator shall supply the two sets of key values described below and obtain the ciphertext values that result from AES encryption of an all zeros plaintext using the given key values and an IV of all zeros. The first set of keys shall have 128 128-bit keys, and the second shall have 256 256-bit keys. Key_i in each set shall have the leftmost i bits be ones and the rightmost N-i bits be zeros, for i in [1, N]. To test the decrypt functionality, the evaluator shall supply the two sets of key and ciphertext value pairs described below and obtain the plaintext value that results from decryption of the given ciphertext using the given key values and an IV of all zeros. The first set of key/ciphertext pairs shall have 128 128-bit key/ciphertext pairs, and the second set of key/ciphertext pairs shall have 256 256-bit pairs. Key_i in each set shall have the leftmost i bits be ones and the rightmost N-i bits be zeros for i in [1, N]. The ciphertext value in each pair shall be the value that results in an all zeros plaintext when decrypted with its corresponding key.

312 To test the encrypt functionality, the evaluator shall supply the set of 128 plaintext values described below and obtain the two ciphertext values that result from encryption of the given plaintext using a 128-bit key value of all zeros and using a 256 bit key value of all zeros, respectively, and an IV of all zeros. Plaintext value i in each set shall have the leftmost bits be ones and the rightmost 128-i bits be zeros, for i in [1, 128]. To test the decrypt functionality, the evaluator shall perform the same test as for encrypt, using ciphertext values of the same form as the plaintext in the encrypt test as input.

313 Test 2: Multi-Block Message Test

314 The evaluator shall test the encrypt functionality by encrypting an i-block message where 1 less-than i less-than-or-equal to 10. For each i the evaluator shall choose a key, IV, and plaintext message of length i blocks and encrypt the message, using the mode to be tested, with the chosen key. The ciphertext shall be compared to the result of encrypting the same plaintext message with the same key and IV using a known good implementation. The evaluator shall also test the decrypt functionality by decrypting an i-block message where 1 less-than i less-than-or-equal to 10. For each i the evaluator shall choose a key and a ciphertext message of length i blocks and decrypt the message, using the mode to be tested, with the chosen key. The plaintext shall be compared to the result of decrypting the same ciphertext message with the same key using a known good implementation.

315 Test 3: Monte-Carlo Test

316 For AES-CTR mode perform the Monte Carlo Test for ECB Mode on the encryption engine of the counter mode implementation. There is no need to test the decryption engine.

317 The evaluator shall test the encrypt functionality using 200 plaintext/key pairs. 100 of these shall use 128 bit keys, and 100 of these shall use 256 bit keys. The plaintext values shall be 128-bit blocks. For each pair, 1000 iterations shall be run as follows:

For AES-ECB mode

```
# Input: PT, Key

for i = 1 to 1000:

   CT[i] = AES-ECB-Encrypt(Key, PT)

   PT = CT[i]
```

318 The ciphertext computed in the 1000th iteration is the result for that trial. This result shall be compared to the result of running 1000 iterations with the same values using a known good implementation.

Findings:	AES-CTR mode with 128-bit and 256-bit keys is claimed for SSH functionality. CAVP C1651 has the appropriate certificates for the claimed platforms for this mode and key sizes: https://csrc.nist.gov/projects/cryptographic-algorithm-validation-program/details?product=12239

319 If "invoke platform-provided" is selected, the evaluator confirms that SSH connections are only successful if appropriate algorithms and appropriate key sizes are configured. To do this, for each listening SSH socket connection on the TOE, the evaluator configures an SSH client to connect with an invalid cryptographic algorithm and key-size. The evaluator observes that the connection fails. Likewise, for initiated connection, the evaluator configures a listening SSH socket on the remote server that accepts only invalid cryptographic algorithms and keys. The evaluator observes that the connection fails.

Findings:	The ST does not claim “invoke platform provided” functionality.

6.1.2 FCS_SSH_EXT.1 SSH Protocol

6.1.2.1 TSS

320 The evaluator will ensure that the selections indicated in the ST are consistent with selections in the dependent components.

Findings:	The evaluator verified that the selections are consistent with the components: 5656 claimed to support ECDSA based public key algorithms. 6668 claimed to support HMAC SHA2 message digest algorithms.

6.1.3 FCS_SSHS_EXT.1 SSH Protocol – Server

6.1.3.1 FCS_SSHS_EXT.1.1

6.1.3.1.1 TSS

321 **TD0420 -** The evaluator will check to ensure that the TSS contains a description of the public key algorithms that are acceptable for use for authentication, that this list conforms to FCS_SSHS_EXT.1.4, and ensure that password-based authentication methods, if supported, are described.

Findings:	Section 6.2.7 of the [ST] specifies that the TOE supports ssh-rsa, ecdsa-sha2-nistp256, ecdsa-sha2-nistp384 and password based authentication methods. This conforms to the selections in FCS_SSHS_EXT.1.4.

6.1.3.1.2 Tests

322 Test 1: The evaluator will, for each public key algorithm supported, show that the TOE supports the use of that public key algorithm to authenticate a user connection from an SSH client. Any configuration activities required to support this test shall be performed according to instructions in the guidance documentation.

High-Level Test Description
Configure the user to be permitted to use the claimed public key algorithm. Load the public key half into the TOE. Log into the TOE with the correct user using the private key half and show it is successful. Repeat for each claimed public key algorithm.
PASS

323 Test 2: The evaluator shall choose one public key algorithm supported by the TOE. The evaluator shall generate a new key pair for that algorithm without configuring the TOE to recognize the public key for authentication. The evaluator shall use an SSH client to attempt to connect to the TOE with the new key pair and demonstrate that authentication fails.

High-Level Test Description
Building on the previous test case, generate a new ssh-rsa key pair. Log into the TOE with the correct user using the newly generated private key half and show it is not successful.
PASS

324 **TD0420 -** Test 3 [conditional]: Using the guidance documentation, the evaluator will configure the TOE to perform password-based authentication on a client, and demonstrate that a user can be successfully authenticated by the TOE using a password as an authenticator.

High-Level Test Description
Log into the TOE using a username and known-good password and show it is successful.
PASS

325 **TD0420 -** Test 4 [conditional]: The evaluator shall use an SSH client, enter an incorrect password to attempt to authenticate to the TOE, and demonstrate that the authentication fails.

High-Level Test Description
Log into the TOE using a username and a-bad password and show it is NOT successful.
PASS

6.1.3.2 FCS_SSHS_EXT.1.2

6.1.3.2.1 TSS

326 The evaluator will check that the TSS describes how large packets in terms of RFC 4253 are detected and handled.

Findings:	Section 6.2.7 of the [ST] specifies that SSH packets greater than 256KB are automatically dropped.

6.1.3.2.2 Tests

327 The evaluator will demonstrate that if the TOE receives a packet larger than that specified in this component, that packet is dropped.

High-Level Test Description
Send a packet from the SSH client to the TOE SSH server slightly smaller than the claimed maximum and show that the TOE accepts the packet. Send a packet from the SSH client to the TOE SSH server slightly larger than the defined maximum and show the TOE drops the packets and terminates the connection.
PASS

6.1.3.3 FCS_SSHS_EXT.1.3

6.1.3.3.1 TSS

328 The evaluator will check the description of the implementation of this protocol in the TSS to ensure that optional characteristics are specified, and the encryption algorithms supported are specified as well. The evaluator will check the TSS to ensure that the encryption algorithms specified are identical to those listed for this component.

Findings:	Section 6.2.7 of the [ST] specifies that encryption algorithms aes128-ctr, aes256-ctr, aes128-cbc, aes256-cbc, aes128- gcm@openssh.com, and aes256-gcm@openssh.com are supported. This list is identical to the selections made in FCS_SSHS_EXT.1.3.

6.1.3.3.2 Guidance Documentation

329 The evaluator will also check the guidance documentation to ensure that it contains instructions on configuring the TOE so that SSH conforms to the description in the TSS (for instance, the set of algorithms advertised by the TOE may have to be restricted to meet the requirements).

Findings:	Section 3.4.1 of the [SUPP] provides the necessary information needed to configure the SSH server to meet the requirements. The list of options covers all of the parameters in the SFR elements. A pointer to the sshd(8) ensures administrators are given the authoritative information on configuring the server.

6.1.3.3.3 Tests

330 Test 1: The evaluator will initiate an SSH connection using each of the encryption algorithms specified by the requirement. It is sufficient to observe (on the wire) the successful negotiation of the algorithm to satisfy the intent of the test.

High-Level Test Description
Connect using each of the claimed ciphersuites and show they are successful in turn.
PASS

331 Test 2: The evaluator will configure an SSH client to only propose the 3des-cbc encryption algorithm and no other encryption algorithms. The evaluator will attempt to establish an SSH connection from the client to the TOE server and observe that the connection is rejected.

High-Level Test Description
Connect to the TOE over SSH using the 3des-cbc cipher and show it fails to successfully negotiate.
PASS

6.1.3.4 FCS_SSHS_EXT.1.4

6.1.3.4.1 TSS

332 The evaluator will check the description of the implementation of this protocol in the TSS to ensure that optional characteristics are specified, and the public key algorithms supported are specified as well. The evaluator will check the TSS to ensure that the public key algorithms specified are identical to those listed for this component.

Findings:	Section 6.2.7 of the [ST] specifies that the TOE supports ssh-rsa, ecdsa-sha2-nistp256, ecdsa-sha2-nistp384 public key algorithms. This conforms to the selections in FCS_SSHS_EXT.1.4.

6.1.3.4.2 Guidance Documentation

333 The evaluator will also check the guidance documentation to ensure that it contains instructions on configuring the TOE so that SSH conforms to the description in the TSS (for instance, the set of algorithms advertised by the TOE may have to be restricted to meet the requirements).

Findings:	Section 3.4.1 of the [SUPP] provides the necessary information needed to configure the SSH server to meet the requirements. The list of options covers all of the parameters in the SFR elements. A pointer to the sshd(8) ensures administrators are given the authoritative information on configuring the server.

6.1.3.4.3 Tests

334 The Test 1: Using an appropriately configured client, the evaluator will establish an SSH connection using each of the public key algorithms specified by the requirement to authenticate. It is sufficient to observe (on the wire) the successful negotiation of the algorithm to satisfy the intent of the test.

NOTE:	Done in FCS_SSHS_EXT.1.1 Test 1.

335 Test 2: The evaluator will configure an SSH client to propose only the ssh-dsa public key algorithm and no other public key algorithms. Using this client, the evaluator will attempt to establish an SSH connection to the TOE and observe that the connection is rejected.

High-Level Test Description
Configure the user to transmit a ssh-dsa key. Load the public key half into the TOE's key store. Log into the TOE with the correct user using the private key half and show it fails.
PASS

6.1.3.5 FCS_SSHS_EXT.1.5

6.1.3.5.1 TSS

336 **This entire section modified by TD0446**

337 The evaluator will check the TSS to ensure that it lists the supported data integrity algorithms, and that that list corresponds to the list in this component.

Findings:	Section 6.2.7 of the [ST] specifies that the TOE supports hmac-sha1, hmac-sha1-96, hmac-sha2-256, and hmac-sha2-512 and implicit (for GCM ciphers) data integrity algorithms. This conforms to the selections in FCS_SSHS_EXT.1.5.

6.1.3.5.2 Guidance Documentation

338 **This entire section modified by TD0446**

339 The evaluator will also check the guidance documentation to ensure that it contains instructions to the administrator on how to ensure that only the allowed data integrity algorithms are used in SSH connections with the TOE (specifically, that the "none" MAC algorithm is not allowed).

Findings:	Section 3.4.1 of the [SUPP] provides the necessary information needed to configure the SSH server to meet the requirements. The list of options covers all of the parameters in the SFR elements. A pointer to the sshd(8) ensures administrators are given the authoritative information on configuring the server. In section 3.4.1 of the [SUPP], the MAC set does not include the "none" algorithm.

6.1.3.5.3 Tests

340 **This entire section modified by TD0446**

341 Test 1: Using an appropriately configured client, the evaluator will establish a SSH connection using each of the integrity algorithms, except "implicit", specified by the requirement. It is sufficient to observe (on the wire) the successful negotiation of the algorithm to satisfy the intent of the test.

High-Level Test Description
Using each of the defined integrity algorithms (and a supported ciphersuite permitting its use), show that the algorithm is supported.
PASS

342 Test 2: The evaluator will configure an SSH client to only allow the “none” MAC algorithm. Using this client, the evaluator will attempt to connect to the TOE and observe that the attempt fails.

Note: To ensure the proposed MAC algorithm is used, the evaluator shall ensure a non-aes*- gcm@openssh.com encryption algorithm is negotiated while performing this test.

High-Level Test Description
Using the ‘none’ integrity algorithms (and a supported ciphersuite permitting its use), show that the algorithm is NOT supported.
PASS

343 Test 3: The evaluator will configure an SSH client to only allow the hmac- md5 MAC algorithm. using this client, the evaluator will attempt to connect to the TOE and observe that the attempt fails.

High-Level Test Description
Using the hmac-md5 integrity algorithms (and a supported ciphersuite permitting its use), show that the algorithm is NOT supported.
PASS

6.1.3.6 FCS_SSHS_EXT.1.6

6.1.3.6.1 TSS

344 The evaluator will check the TSS to ensure that it lists the supported key exchange algorithms, and that that list corresponds to the list in this component.

Findings:	Section 6.2.7 of the [ST] specifies that the TOE supports the diffie-hellman-group14-sha1 key exchange algorithm. This conforms to the selections in FCS_SSHS_EXT.1.6.

6.1.3.6.2 Guidance Documentation

345 The evaluator will also check the guidance documentation to ensure that it contains instructions to the administrator on how to ensure that only the allowed key exchange algorithms are used in SSH connections to the TOE.

Findings:	Section 3.4.1 of the [SUPP] provides the necessary information needed to configure the SSH server to meet the requirements. The list of options covers all of the parameters in the SFR elements. A pointer to the sshd(8) ensures administrators are given the authoritative information on configuring the server.

6.1.3.6.3 Tests

346 Test 1: For each of the allowed key exchange methods, the evaluator will configure an SSH client to propose only it and attempt to connect to the TOE and observe that each attempt succeeds.

High-Level Test Description
Using each of the defined key exchange algorithms show that the algorithm is supported.
PASS

347 Test 2: The evaluator shall configure an SSH client to only allow the diffie-hellman-group1-sha1 key exchange. The evaluator shall attempt to connect from the SSH client to the SSH Server and observe that the attempt fails.

High-Level Test Description
Using the diffie-hellman-group1-sha1 key exchange algorithm show that the algorithm is NOT supported.
PASS

6.1.3.7 FCS_SSHS_EXT.1.7

6.1.3.7.1 Tests

348 **TD0331 -** Test 1: The evaluator will configure the TOE to create a log entry when a rekey occurs. The evaluator will connect to the TOE with an SSH client and cause a rekey to occur according to the selection(s) in the ST, and subsequently the evaluator uses available methods and tools to verify that rekeying occurs. This could be done, e.g., by checking that a corresponding audit event has been generated by the TOE, if the TOE supports auditing of rekey events.

High-Level Test Description
Show that when the TOE reaches its rekey limits, the TOE will perform a rekey operation and that the rekey action is capable of being logged.
PASS