The Bugs Framework (BF) "Hands-On"

Software developer's and tester's "Best Friend"

Irena Bojanova, NIST
Paul Black, NIST

I. BF: Buffer Overflow (BOF)

1. BOF Taxonomy

Definition

The software accesses through an array a memory location that is outside the boundaries of that array.

Related CWEs, SFP and ST:

- ✓ Related CWEs are CWE-119, CWE-120, CWE-121, CWE-122, CWE-123, CWE-124, CWE-125, CWE-126, CWE-127, CWE-786, CWE-787, CWE-788.
- ✓ The related SFP cluster is SFP8 Faulty Buffer Access under Primary Cluster: Memory Access.
- ✓ The corresponding ST is the Buffer Overflow Semantic Template.

The causes, attributes and consequences of the BOF class are depicted and explained below.

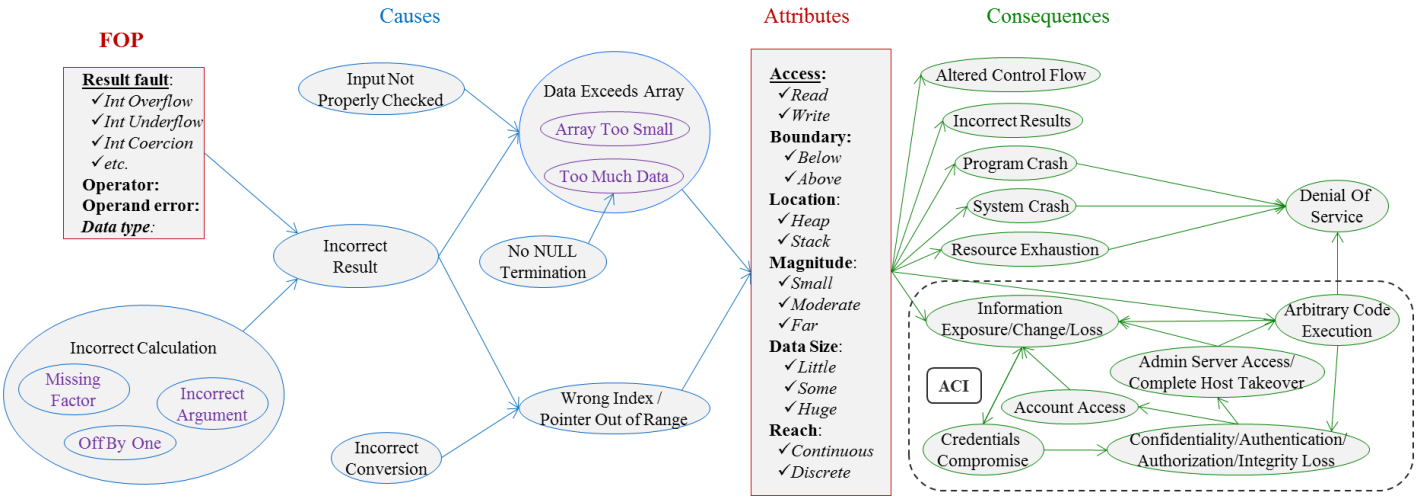
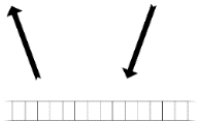


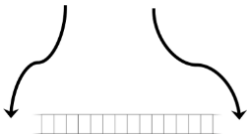
Figure 1. Buffer Overflow (BOF) class.

Attributes
The attributes for buffer overflows are as follows:

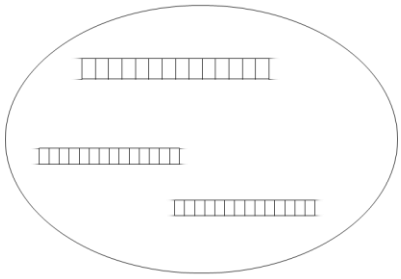
- **Access** – Read, Write.

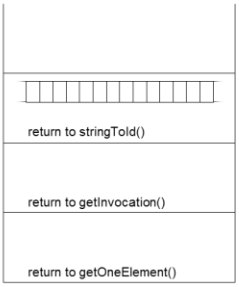


- **Boundary** – Below, Above.
  Which end of the array is violated. Synonyms for boundary are side, bound or end. The terms before, under or lower may be used instead of below. The terms after, over or upper may be used instead of above. The term outside indicates that the boundary is unknown or it doesn't matter.



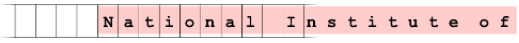
- **Location** – Heap, Stack.
  What part of memory the array is allocated in. It may matter since violations in the stack may affect program execution flow, while violations in the heap typically only affect data values. Other compilers and operating system may have other locations that are significant. For instance, other locations are BSS (uninitialized data), Data (initialized) and Code (text).



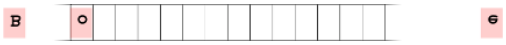


- **Magnitude** – Small, Moderate, Far.
  How far outside the boundary the violation extends. Small means just barely outside, e.g. one to a few bytes, moderate means is from eight bytes to dozens and far is hundreds, thousands or more.

- **Data Size** – Little, Some, Huge.
  How much data is accessed beyond the boundary. As in magnitude, these distinctions are important in some cases. For instance, Heartbleed might not have been a severe problem if it just exfiltrated a little data. The fact that it may exfiltrate a huge amount of data greatly increases the chance that very important information will be leaked.



- **Reach** – Continuous, Discrete.
  Whether the violating access was preceded by consecutive accesses of elements within the array (continuous) or the violation was just accessing outside of the array (discrete). Synonym for reach is span. Typically string accesses or array copies handle a continuous set of array elements, while a vagrant array index only reads or writes one element.



Causes

- **Data Exceeds Array**
  - ✓ Array Too Small
    The array was allocated smaller than it should have been. This may occur because the programmer leaves out a factor, like the size of a header, uses the wrong variable, or forgets room for a null to terminate a string.
  - ✓ Too Much Data
    More data is accessed than was anticipated. This may occur because the string is not NULL terminated or the amount of data is calculated differently than the size of the buffer (e.g. heartbleed).

Wrong Index / Pointer out of Range

Contributing Causes

- ✓ Data Exceeds Array may be caused by an Input [that is] Not Checked Properly [REF] or by an Incorrect Calculation [REF]. The specific cases of Data Exceeds Array are Array Too Small and Too Much Data. Too Much Data may be caused by No NULL Termination [REF]. Wrong Index [or] Pointer Out of Range may be caused by Incorrect Calculation, too, or by an Incorrect Conversion [REF].
- ✓ The specific cases of Incorrect Calculation are Missing Factor, Incorrect Argument, Off By One, Integer Coercion, Integer Overflow Wrap-around and Integer Underflow.

Consequences
The graph of consequences shows what could happen due to the fault. The ACI cluster of consequences is the same in all classes where it appears. “Resource Exhaustion” refers to Memory and CPU.

Sites
BOF may occur at the use of [ ] or the use of unary * operator with arrays in the C language. Sites also include the use of many string library functions, such as strcpy() or strcat().

2. BOF Exercises

Use BF BOF (fig. 1) to describe known software vulnerabilities or to identify gaps in existing repositories:

1) Ghost: BOF → CVE-2015-0235
2) Chrome: BOF → CVE-2010-1773
3) CWE gaps: BOF → Refactoring CWEs

2.1. BOF: Exercise 1 (Ghost) – CVE-2015-0235

Create a BF description of CVE-2015-0235:
1. Examine the listed below CVE description, references, and source code excerpts with bug and fix.
2. Analyze the gathered information and come up with a BF description utilizing the BOF taxonomy (causes, attributes, and consequences).

CVE-2015-0235 (Ghost): "Heap-based buffer overflow in the __nss_hostname_digits_dots function in glibc 2.2, and other 2.x versions before 2.18, allows context-dependent attackers to execute arbitrary code via vectors related to the (1) gethostbyname or (2) gethostbyname2 function, aka GHOST." [6]

References

[6] The MITRE Corporation, CVE Common Vulnerabilities and Exposures, CVE-2015-0235.
[7] Openwall, bringing security into open environment, Qualys Security Advisory CVE-2015-0235.
[8] Qualys Security Advisory CVE-2015-0235.

BOF: Exercise 1 – Source Code

Code With Bug

```
1  /* calculate size incorrectly*/
2  size_needed = (sizeof (*host_addr)+ sizeof (*h_addr_ptrs)
                  + strlen (name) + 1);
3
4  host_addr = (host_addr_t *) *buffer;
5  h_addr_ptrs = (host_addr_list_t *)((char *) host_addr
                  + sizeof (*host_addr));
6  hostname = (char *) h_addr_ptrs + sizeof (*h_addr_ptrs);
7  resbuf->h_name = strcpy (hostname, name);
```

Code With Fix

```
1  /* calculate size incorrectly*/
2  size_needed = (sizeof (*host_addr) + sizeof (*h_addr_ptrs)
                  + sizeof (*h_alias_ptr) + strlen (name) + 1);
3
4  host_addr = (host_addr_t *) *buffer;
5  h_addr_ptrs = (host_addr_list_t *)((char*) host_addr
                  + sizeof (*host_addr));
6  hostname = (char*) h_addr_ptrs + sizeof (*h_addr_ptrs);
7  resbuf->h_name = strcpy (hostname, name);
```

BOF: Exercise 1 – Analysis (use if you do not have a way to explore the references)
The following analysis is based on information in [6, 7, 8].

- The number of bytes that can be overwritten is sizeof (char *), which is 4 bytes on a 32 bit machine, and 8 bytes on a 64 bit machine.
- In a calculation of the size needed to store certain data, the size of a char pointer is missing, resulting in array too small.
- Buffer over write is done by strcpy (continuous reach).
- Qualys developed an attack on the Exim mail server, exploiting this vulnerability, as proof of concept.
- This attack uses an initial buffer overwrite to enlarge the number in the size field of a portion of memory that is available for the next allocation.
- This modification enables a subsequent overwrite that enables write-anything-anywhere, which in turn enables overwriting Exim's Access Control Lists, which in turn enables arbitrary code execution.

2.2. BOF: Exercise 2 (Chrome) – CVE-2010-1773

Create a BF description of CVE-2010-1773:
1. Examine the listed below CVE description, references, and source code excerpts with bug and fix.
2. Analyze the gathered information and come up with a BF description utilizing the BOF taxonomy.

CVE-2010-1773 (Chrome WebCore): "Off-by-one error in the toAlphabetic function in rendering/RenderListMarker.cpp in WebCore in WebKit before r59950, as used in Google Chrome before 5.0.375.70, allows remote attackers to obtain sensitive information, cause a denial of service (memory corruption and application crash), or possibly execute arbitrary code via vectors related to list markers for HTML lists, aka rdar problem 8009118." [9]

References

[9] The MITRE Corporation, CVE Common Vulnerabilities and Exposures, CVE-2010-1773.
[10] Robin Gandhi, Buffer Overflow Semantic template CVE-2010-1773.
[11] Tracker, Issue 44955.
[12] chromium, Diff of /branches/WebKit/375/WebCore/rendering/RenderListMarker.cpp. Revision 48099.
[13] chromium, Contents of /branches/WebKit/375/WebCore/rendering/RenderListMarker.cpp. Revision 44321.
[14] chromium, Contents of /branches/WebKit/375/WebCore/rendering/RenderListMarker.cpp. Revision 48100.
[15] webkit, Fix for Crash in WebCore::toAlphabetic() while running MangleMe -and corresponding- https://bugs.webkit.org/show_bug.cgi?id=39508. Reviewed by Darin Adler.
[16] Hat Bugzilla – Bug 596500- (CVE-2010-1773) CVE-2010-1773 WebKit: off-by-one memory read out of bounds vulnerability in handling of HTML lists.

BOF: Exercise 2 – Source Code

Code With Bug

```
1 if (type == AlphabeticSequence)
2 {
3     while ((numberShadow /= sequenceSize) > 0)
4     {
5         letters[lettersSize  - ++length] = sequence[numberShadow % sequenceSize - 1];
6     }
7 }
```

Code With Fix

```
1 if (type == AlphabeticSequence)
2 {
3     while ((numberShadow /= sequenceSize) > 0)
4     {
5         --numberShadow;
6         letters[lettersSize - ++length] = sequence[numberShadow % sequenceSize];
7     }
8 }
```

BOF: Exercise 2 – Analysis (use if you do not have a way to explore the references)
The following analysis is based on information in [9-16].

- The software reads in a loop from an array, where the sequence of indices of array elements read is neither necessarily monotonic nor necessarily having a fixed distance between consecutive elements.
- That index should be the remainder obtained by dividing an integer by an integer.
- The software subtracts 1 from that remainder, which is wrong, and can result in the index being equal to -1, leading to reading from an address that is below the beginning of the array by 1.
- Consequences are mentioned in [10], and [16] includes "An off by one memory read out of bounds issue exists in WebKit's handling of HTML lists. Visiting a maliciously crafted website may lead to an unexpected application termination or the disclosure of the contents of memory."

2.3. BOF: Exercise 3 (Refactoring CWEs)

Examine the CWEs listed below and fill in the table on the next page.

CWE-120: Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')
The program copies an input buffer to an output buffer without verifying that the size of the input buffer is less than the size of the output buffer, leading to a buffer overflow.
CWE-121: Stack-based Buffer Overflow
A stack-based buffer overflow condition is a condition where the buffer being overwritten is allocated on the stack (i.e., is a local variable or, rarely, a parameter to a function).
CWE-122: Heap-based Buffer Overflow
… a buffer overflow, where the buffer that can be overwritten is allocated in the heap portion of memory, generally meaning that the buffer was allocated using a routine such as malloc().

CWE-123: Write-what-where Condition
Any condition where the attacker has the ability to write an arbitrary value to an arbitrary location, often as the result of a buffer overflow.
CWE-124: Buffer Underwrite (’Buffer Underflow')
The software writes to a buffer using an index or pointer that references a memory location prior to the beginning of the buffer.
CWE-125: Out-of-bounds Read
The software reads data past the end, or before the beginning, of the intended buffer.
CWE-126: Buffer Over-read
The software reads from a buffer using buffer access mechanisms such as indexes or pointers that reference memory locations after the targeted buffer.
CWE-127: Buffer Under-read
The software reads from a buffer using buffer access mechanisms such as indexes or pointers that reference memory locations prior to the targeted buffer.

CWE-786: Access of Memory Location Before Start of Buffer
The software reads or writes to a buffer using an index or pointer that references a memory location prior to the beginning of the buffer.
CWE-787: Out-of-bounds Write
The software writes data past the end, or before the beginning, of the intended buffer.
CWE-788: Access of Memory Location After End of Buffer
The software reads or writes to a buffer using an index or pointer that references a memory location after the end of the buffer.
CWE-805: Buffer Access with Incorrect Length Value
The software uses a sequential operation to read or write a buffer, but it uses an incorrect length value that causes it to access memory that is outside of the bounds of the buffer.
CWE-823: Use of Out-of-range Pointer Offset
The program performs pointer arithmetic on a valid pointer, but it uses an offset that can point outside of the intended range of valid memory locations for the resulting pointer.

Please put a check in the BOF attributes for each CWE.

Note: You can leave an attribute blank (or check both) if the CWE is "unknown", "don't care", or "both". We did CWE-120 as an example.

Attributes	Access		Boundary		Location		Reach	
	read	write	lower	upper	heap	stack	continuous	discrete
120		✓		✓	✓	✓	✓	
121								
122								
123								
124								
125								
126								
127								
786								
787								
788								
805								
823								

II. BF: Cryptography (CRY)

1. Taxonomy

- Definition:
  The software does not properly encrypt/decrypt, verify, or manage keys for data (that has) to be securely stored/transferred.

  _ENC: The software does not properly transform sensitive data (plaintext) into unintelligible form (ciphertext) using cryptographic algorithm and key(s).
  _DEC: The software does not properly transform ciphertext into plaintext using cryptographic algorithm and key(s).
  _VRF: The software does not properly sign message, check and prove sender, or assure message is not altered.
  _KMN: The software does not properly generate, store, distribute, use, or destroy cryptographic keys (keying material).

- Related CWEs and SFPs:
  - ✓ CWEs related to CRY are CWE-311, CWE-325, CWE-327, CWE- 261, CWE-322, CWE-323, CWE-324, CWE-326, CWE-347, CWE-312 (incl. 313-318), CWE-256, CWE-257, CWE-295, CWE-296, CWE-321, CWE-329, CWE-780
  - ✓ Related SFPs SFP 17.1 and SFP 17.2 under Primary Cluster: Cryptography.

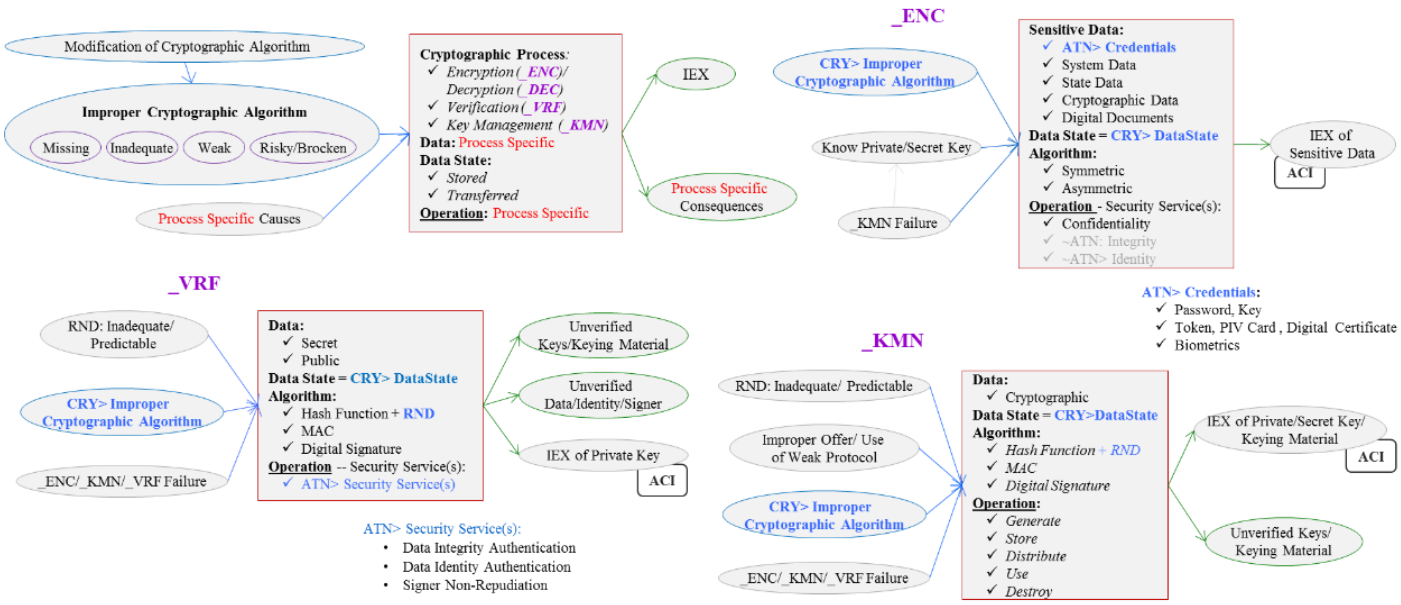


Figure 2. Cryptography (CRY) class.

2. CRY Exercises

Use BF CRY (Fig. 2) to describe known software vulnerabilities or to identify gaps in existing repositories:

1) FREAK: CRY → CVE-2015-0204, CVE-2015-1637, CVE-2015-1067

2.1. CRY: Exercise 1 (FREAK) – CVE-2015-0204, CVE-2015-1637, CVE-2015-1067

Create a BF description for FREAK – CVE-2015-0204, CVE-2015-1637, CVE-2015-1067:
1. Examine the listed below CVE descriptions, references, and source code excerpts with bug and fix.
2. Analyze the gathered information and come up with a BF description utilizing the CRY taxonomy.

CVE-2015-0204: "The ssl3_get_key_exchange function in s3_clnt.c in OpenSSL before 0.9.8zd, 1.0.0 before 1.0.0p, and 1.0.1 before 1.0.1k allows remote SSL servers to conduct RSA-to-EXPORT_RSA downgrade attacks and facilitate brute-force decryption by offering a weak ephemeral RSA key in a noncompliant role, related to the "FREAK" issue. NOTE: the scope of this CVE is only client code based on OpenSSL, not EXPORT_RSA issues associated with servers or other TLS implementations." [17]

CVE-2015-1637: "Schannel (aka Secure Channel) in Microsoft Windows Server 2003 SP2, Windows Vista SP2, Windows Server 2008 SP2 and R2 SP1, Windows 7 SP1, Windows 8, Windows 8.1, Windows Server 2012 Gold and R2, and Windows RT Gold and 8.1 does not properly restrict TLS state transitions, which makes it easier for remote attackers to conduct cipher-downgrade attacks to EXPORT_RSA ciphers via crafted TLS traffic, related to the "FREAK" issue, a different vulnerability than CVE-2015-0204 and CVE-2015-1067." [18]

CVE-2015-1067: "Secure Transport in Apple iOS before 8.2, Apple OS X through 10.10.2, and Apple TV before 7.1 does not properly restrict TLS state transitions, which makes it easier for remote attackers to conduct cipher-downgrade attacks to EXPORT_RSA ciphers via crafted TLS traffic, related to the "FREAK" issue, a different vulnerability than CVE-2015-0204 and CVE-2015-1637." [19]

References

[17] The MITRE Corporation, CVE Common Vulnerabilities and Exposures, CVE-2015-0204.
[18] The MITRE Corporation, CVE Common Vulnerabilities and Exposures, CVE-2015-1637.
[19] The MITRE Corporation, CVE Common Vulnerabilities and Exposures, CVE-2015-1067.
[20] Rob Heaton, The SSL FREAK vulnerability explained, http://robertheaton.com/2015/04/06/the-ssl-freak-vulnerability
[21] Censys, The FREAK Attack, https://censys.io/blog/freak.
[22] StackExchange, Protecting phone from the FREAK bug, http://android.stackexchange.com/questions/101929/protecting-phone-from-the-freak-bug/101966
[23] GitHub, openssl, https://github.com/openssl/openssl/commit/ce325c60c74b0fa784f5872404b722e120e5cab0?diff=split

CRY: Exercise 1 (FREAK) – Source Code

Client

```
#ifndef OPENSSL_NO_RSA
```

```
if (alg_k & SSL_kRSA) {
```

```
if (alg_k & SSL_kRSA) {
  if (!SSL_C_IS_EXPORT(s->s3->tmp.new_cipher)) {
    al=SSL_AD_UNEXPECTED_MESSAGE;
    SSLerr(SSL_F_SSL3_GET_SERVER_CERTIFICATE,
       SSL_R_UNEXPECTED_MESSAGE);
    goto f_err;
  }
```

```
if ((rsa=RSA_new()) == NULL) {
SSLerr(SSL_F_SSL3_GET_KEY_EXCHANGE,
ERR_R_MALLOC_FAILURE);
```

```
if ((rsa=RSA_new()) == NULL) {
SSLerr(SSL_F_SSL3_GET_KEY_EXCHANGE,
ERR_R_MALLOC_FAILURE);
```

Server

```
case SSL3_ST_SW_KEY_EXCH_B:
alg_k = s->s3->tmp.new_cipher->algorithm_mkey;
if ((s->options & SSL_OP_EPHEMERAL_RSA)
#ifndef OPENSSL_NO_KRB5
 && !(alg_k & SSL_kKRB5)
#endif )
    s->s3->tmp.use_rsa_tmp=1;
else
    s->s3->tmp.use_rsa_tmp=0;
if (s->s3->tmp.use_rsa_tmp
```

```
case SSL3_ST_SW_KEY_EXCH_B:
alg_k = s->s3->tmp.new_cipher->algorithm_mkey;
s->s3->tmp.use_rsa_tmp=0;
if (
```

- If client ciphersuit is non-export then returned by server RSA keys should be also non-export. Therefore, handshake that offers export RSA key (512 bits, which is weak) should be abandoned by client.
- The buggy code includes a handshake that enables accepting a 512-bit RSA key.
- The fix is adding code that checks whether client ciphersuit is non-export and for abandoning the handshake if this is the case.

CRY: Exercise 1 (FREAK) – Analysis (use if you do not have a way to explore the references)
The following analysis is based on information in [17-23].

ClientHello: Client sends plaintext with no sensitive data (set of cipher suites client to use).
→ MITM intercepts that plaintext and changes it to request only Export RSA ciphersuite.

ServerHello: If configured to offer Export RSA, server responds sending its SSL certificate.

Send ServerKeyExchange: Server generates/retrieves 512-bit RSA key-pair, signs public key with its SSL certificate's private key to authenticate it to client, and sends the 512-bit key and signature.
→ MITM is watching exchange.

Receive ServerKeyExchange: Client receives *ServerKeyExchange* (with weak Export RSA key). If there was no bug in OpenSSL, client should have said it did not ask for this and error out, shutting down the attack. However, due to this bug client would accept and use this weak key for the rest of the handshake.

Pre-Master Secret: In both normal RSA and Export RSA, the Master Secret, used for symmetric encryption of all messages in the rest of the connection, is generated using Pre-Master Secret and two nonces (*ClientRandom* and *ServerRandom* sent in plaintext respectively in *ClientHello* and *ServerHello*). Client generates Pre-Master Secret as a random string, encrypts it using server's public key, and sends it to server. However, in Export RSA, server's public key is the 512-bit key from *ServerKeyExchange* rather than from the server's SSL certificate.
→ MITM can factor that weak 512-bit public key to obtain the RSA private key.
(Normal RSA public key is min 1024 bits to render such factoring infeasible, but a 512-bits public key can be factored using $100 of AWS computing time.)
→ With the RSA private key MITM can decrypt Pre-Master Secret, and then use it and the nonces to find the Master Secret (secret key used for symmetric encryption of transmitted data).
→ MITM now can decrypt, read, modify any massage between client and server (passwords, credit cards info, etc.)

Note: For Export RSA, a weaker RSA key-pair (512-bit) is required than required on the SSL certificate. If it was RSA, the client would generate the Pre-Master Secret and encrypt it with server's public key (min 1024-bit) from its SSL certificate.