# Computer Security MT14: Practical 1

A Plaintext-Insertion Attack

Underestimating the powers of the attacker is often at the root of practical attacks on security. An attack which gained some notoriety in 2011 was the so-called "BEAST" attack on SSL and TLS version 1.0 (protocols which we will see in the final chapter of the course), in which malicious javascript gives the attacker something new: the ability to insert some chosen plaintext *inside* the plaintext which they wish to attack. Particularly when paired with the CBC block mode, this allows for linear-time decryption of an arbitrary ciphertext. You will implement an abstract version of the attack.

**Instructions**

**To complete this practical, you should hand in a completed and well-commented Java program `BeastAttack.java`, plus a selection of its output. To qualify for grade `S`, you should make good progress on tasks 1 & 2. You should qualify for an `S+` if your answer to all tasks are complete and efficient, particularly in terms of the number of ciphertexts which your attack requires.**

---

The scenario is as follows. Some client $C$ is communicating with a host $H$, using symmetric encryption with pre-shared keys. The cipher uses 64-bit blocks, and we will write

$$E(\langle b_1, \ldots, b_8 \rangle) = \langle c_1, \ldots, c_8 \rangle$$

to mean that the 8-byte block $\langle b_1, \ldots, b_8 \rangle$ encrypts to $\langle c_1, \ldots, c_8 \rangle$ under the unknown key. The block mode is CBC. We may assume that the cipher cannot be broken directly (and you should *not* try to do so!). The client is attempting to send one piece of highly secret information, a sequence of bytes (padded to a multiple of 8) that we will call

$$\langle m_1, m_2, \ldots, m_{8n} \rangle;$$

we will intercept and block the ciphertext, after which the client will try again and again with the same plaintext. Conveniently, we have placed some malicious software on the client's computer which allows us to add a prefix, of up to 8 bytes, to the plaintext, and we can vary the prefix for every communication the client makes.

Our attack, which comes from the family called *blockwise-adapative chosen plaintext attacks* and is also known as a *chosen-boundary attack*, works as follows. Suppose that we force the 7-byte prefix $\langle 0, 0, 0, 0, 0, 0, 0 \rangle$ onto the plaintext; the first 16 bytes of the ciphertext must therefore be

$$\langle iv_1, \ldots, iv_8, c_1, \ldots, c_8 \rangle$$

where the first eight are the IV and

$$E(\langle iv_1 \oplus 0,\ iv_2 \oplus 0,\ \ldots, iv_7 \oplus 0,\ iv_8 \oplus m_1 \rangle) = \langle c_1, \ldots, c_8 \rangle.$$

After observing $\langle c_1, \ldots, c_8 \rangle$, we can determine $m_1$ as follows:

For each byte $x$,

(i) make the host encrypt the prefix $\langle 0, 0, 0, 0, 0, 0, 0, x \rangle$,

(ii) look only at the second cipher block, and see which one matches $\langle c_1, \ldots, c_8 \rangle$.

The block that matches came from encrypting $\langle 0, 0, 0, 0, 0, 0, 0, m_1 \rangle$.

Then we can repeat, first forcing the prefix $\langle 0, 0, 0, 0, 0, 0 \rangle$ to get a target cipher block which encrypts $\langle iv_1 \oplus 0,\ iv_2 \oplus 0,\ \ldots, iv_6 \oplus 0,\ iv_7 \oplus m_1,\ iv_8 \oplus m_2 \rangle$, then trying all prefixes of the form $\langle 0, 0, 0, 0, 0, 0, m_1, x \rangle$ and looking for a match. The process can continue to recover the entire first message block, and then move onto subsequent blocks, remembering that under CBC each plaintext block is xor-ed with the last cipher block, prior to encryption.

The defence against this attack is the IV: in the simplest form described above, the attack only works if the IV is the same for every message. If the IV is completely unpredictable, the attack cannot be used, because it depends on spotting a match in the cipher blocks, something which will not happen more often than random. But if the IV is variable but predictable (with a reasonable probability of correctness), the attack can be adapted, by xor-ing the predicted IV with the prefix block, cancelling the effect of the IV altogether. The "BEAST" attack works because (in the relevant context) SSL/TLS defaults to using the last cipher block of the previous message as the IV of the next, making it completely predictable.

This practical will have to be run on the lab machines, because you are given executable code which only works correctly on them. You simulate the client, encrypting your chosen prefix and the secret plaintext (which is unique to you), by the executable `/usr/local/practicals/security/encrypt`[1]. The argument to `encrypt` is the prefix, if any, as a string of hexadecimal (no spaces between bytes) and it returns a hexadecimal string which is the ciphertext encrypted by the unknown key. In `/usr/local/practicals/security/` is an incomplete Java program `BeastAttack.java`. It includes a method `callEncrypt()` to launch the `encrypt` executable, as well as to convert its input and output between hex string and `byte[]`.

*General implementation tips*

*In Java, a hexadecimal constant is prefixed by* `0x`*; to output a hexadecimal number, use* `Long.toHexString()` *or* `String.format("%#x",...)`*: the latter prints a leading* `0x` *as well.* `System.out.printf()` *is another alternative.*

*Annoyingly, all of Java's integer data types are signed integers in two's complement, and we will store the message as a sequence of* `Bytes`*. This doesn't affect much because we will never see Java's interpretation of the value of the* `Bytes`*, but be careful if you do* $<$ *or* $>$ *comparisons between them.*

[1] Credit for crypto code: includes software developed by the OpenSSL Project for use in the OpenSSL Toolkit `http://www.openssl.org/`.

In this practical the IV will be different for every use, it is not completely unpredictable. Your first task is to find a way to predict it.

**Task 1**

**Write some code to call `encrypt`, without any prefix, both to determine the length of the plaintext $\langle m_1, \ldots, m_{8n} \rangle$ and to work out how the IV varies on each encryption.**

*Tips for task 1*

*Try running* `encrypt` *multiple times, outputting the first (IV) block of the ciphertext, to see if you can spot any sort of pattern in the results. How do you think it might be generated? You need not reverse-engineer it completely and it is* not *necessary to predict the IV with perfect accuracy: you only need to have a good chance of guessing its value for the next encryption.*

---

**Task 2**

**Adapting the chosen-boundary attack to account for a predicted IV, use it to decrypt the first byte $m_1$ of the plaintext. Then proceed to decrypt the whole of the first 64-bit plaintext block, byte-by-byte. Your answer should be displayed as ASCII text.**

*Tips for task 2*

*It does not matter if you cannot predict the IV exactly. Take a guess, build the appropriate prefix, call* `encrypt`*, and then check whether the ciphertext did have the correct IV block after all. If not, try again with a new predicted IV.*

---

And now you can complete the entire attack.

**Task 3**

**Recover the complete plaintext.**

*Tips for task 3*

*You can repeat what you have done for the first plaintext block to get every block, but you need to account for the xor in the CBC mode.*

*Call* `encrypt` *once with a prefix of 0–7 bytes, noting the cipher block which contains the plaintext block we are attacking, and the previous block as well (so that we can xor away the effect of CBC). Then call* `encrypt` *with blocks of length 8: you only need to observe the second block of the ciphertext.*

---

The "BEAST" attack was demonstrated in 2011, but the idea of blockwise-adaptive chosen plaintexts goes back to 2001-2 and Bard described a sort of abstract "BEAST" attack in 2006. As well as the analysis above, the attack requires some kind of embedded Java tricks

to fool the client into merging their secret data with the attacker's chosen plaintext, prior to encryption. There are a lot of dangers with embedded applets, particularly because data from multiple sources can be pulled together, and their security privileges mixed up.

Although the "BEAST" publication seems difficult to find, there is a readable analysis at `http://www.educatedguesswork.org/2011/09/security_impact_of_the_rizzodu.html` and a draft paper by Duong and Rizzo at `https://bug665814.bugzilla.mozilla.org/attachment.cgi?id=540839`. Other attacks have followed "BEAST", making use of an attacker's ability to insert plaintext into the target. Notable examples include "CRIME" (2012) and "BREACH" (2013), which simply measure the length of the ciphertext: it leaks information when the plaintext is compressed before encryption.

`adk@cs.ox.ac.uk`, MT 2014