Bar Mitzvah Attack

Breaking SSL with a 13-year old RC4 Weakness

Abstract

RC4 is the most popular stream cipher in the world. In fact, as of March 2015, RC4 is estimated to protect as much as 30% of SSL traffic, likely amounting to billions of TLS connections every day. Yet it suffers a critical – and long known – weakness known as the Invariance Weakness. In this paper we will revisit the Invariance Weakness – a 13-year old vulnerability of RC4 that is based on huge classes of RC4 weak keys, which was first published in the FMS paper in 2001. We will show how this vulnerability can be used to mount partial plaintext recovery attacks on SSL-protected data, when RC4 is the cipher of choice, for recovering the LSBs of as many as 100 bytes from the encrypted stream.

As opposed to BEAST, POODLE, CRIME and other attacks on SSL that were published in recent years, including the Royal Holloway Attack on the usage of RC4, a new attack based upon the Invariance Weakness does not rely on aggregation of small fragments of plaintext information, but on a "hit", a rare event that causes a significant leakage to occur. We show how this unique characteristic can be used to attack SSL in new scenarios, including the first practical attack on SSL that does not require an active Man-in-the-Middle. Furthermore, the new attack is not limited to recovery of temporal session tokens, but can be used to steal parts of permanent secret data such as account credentials and credit card numbers when delivered over HTTPS. Another variant of the attack recovers a significant part of a secret with small but non-negligible probability, even if that was transmitted only once over the SSL connection.

This paper will describe the Invariance Weakness in detail, explain its impacts, and recommend some mitigating actions.

Introduction

TLS

The Protocol

TLS is the most widely used secure communications protocol on the Internet today. Starting life as SSL, the protocol was adopted by the IETF and specified as an RFC standard under the name of TLS 1.0 [1]. It has since evolved through TLS 1.1 [2] to the current version TLS 1.2 [3]. TLS 1.3 is, as of March 2015, in draft [4]. Various other RFCs define additional TLS cryptographic algorithms and extensions.

SSL is currently used for securing a wide variety of application-level traffic: It serves, for example, as the basis of the HTTPS protocol for encrypted web browsing, it is used in conjunction with IMAP or SMTP to cryptographically protect email traffic, and it is a popular tool to secure communication with embedded systems, mobile devices, and in payment systems.

SSL strives to fulfill two major goals: 1) allow two parties to authenticate each other, and 2) secure the communication between the two. In many SSL deployments, particular secure web browsing, the authentication is one-way, meaning that only the client (browser) authenticates the server (web application), but not vice versa.

SSL sessions consist of two phases: In the SSL Handshaking Protocol the client authenticates the server, the server (optionally) authenticates the client and both establish cryptographic session keys, ready to protect the communication. In the Record Protocol the parties use the established session keys and symmetric key cryptography to encrypt (e.g., using AES block cipher or RC4 stream cipher) and authenticate (e.g., using HMAC algorithms) to build a secure channel for application-layer data. The parties can choose between many different authentication and encryption algorithms for the Record Protocol, essentially divided into the following classes:

- Block Cipher (CBC mode of operation) + HMAC
- Stream Cipher (RC4) + HMAC
- Authenticated-Encryption using block cipher (GCM/CCM mode of operation)

SSL Protocol Weaknesses

In the last couple of years several significant vulnerabilities have been discovered in the SSL protocol, particularly in the most commonly used variants, AES-CBC and RC4. The AES-CBC variant has seen significant cryptanalysis (padding oracle attacks [5], BEAST [6], Lucky 13 [7], TIME [16], and POODLE [15]). And in 2013 AlFardan et-al published an analysis [8] of the RC4 mode, which showed how to mount an attack that recovers data transmitted over a SSL/RC4 connection. The attack was based on some of the many know weaknesses of RC4, in particular the significant statistical biases in its first output bytes, and the weaker statistical biases in the RC4 keystream.

Implementation Weaknesses

In recent years SSL implementations have received significant attention and scrutiny from security researchers, and this has resulted in steady discovery of new vulnerabilities (and patching, with or without disclosure). In 2014 several dozen vulnerabilities were discovered in the OpenSSL library (the most popular implementation of SSL), Heartbleed being the most severe one in that it allows an attacker to dump memory segments from the SSL server, in many cases exposing secret key information. Vulnerabilities in several other implementations were published as well (e.g., CVE-2014-6321 in Microsoft SChannel).

On RC4

The Stream Cipher

The 4-line Stream Cipher Rivest Cipher 4 (RC4) is one of the simplest cryptographic algorithms, implementing a pseudo-random generator that is used to implement a stream cipher. The internal state of RC4 includes a permutation S of [0, 1, .., 255] and two indices i and j in this permutation.
In the key scheduling phase (KSA) an L-byte long RC4 key (for L varying between 5 and 256) is used to construct the initial state permutation S_0. In the encryption phase, RC4 PRGA, which is initialized with the resultant S_0 is used to generate a stream of pseudo-random bytes, denoted as the keystream. Like most of the Stream Ciphers, these pseudo-random bytes are XOR-ed with the plaintext bytes to generate the ciphertext bytes.

The building blocks of RC4 are described below.

KSA(K)		***PRGA(S_0)***

j = 0 *S = [0, 1, 2, …, 255]* *for i = 0..255* *j = (j + S[i] + K[i mod L])* *S[i] ↔ S[j]*	*All operations are done mod 256*	*i, j = 0, 0* *S = S_0* *while bytes are needed:* *i = i + 1* *j = j + S[i]* *S[i] ↔ S[j]* *Emit S[S[i]+S[j]]*

Known for its simplicity and for its respected author, RC4 gained considerable popularity. And given its impressive performance, being able to encrypt and decrypt almost two times faster than AES, for many years it was considered to be the default stream cipher.

Security of RC4

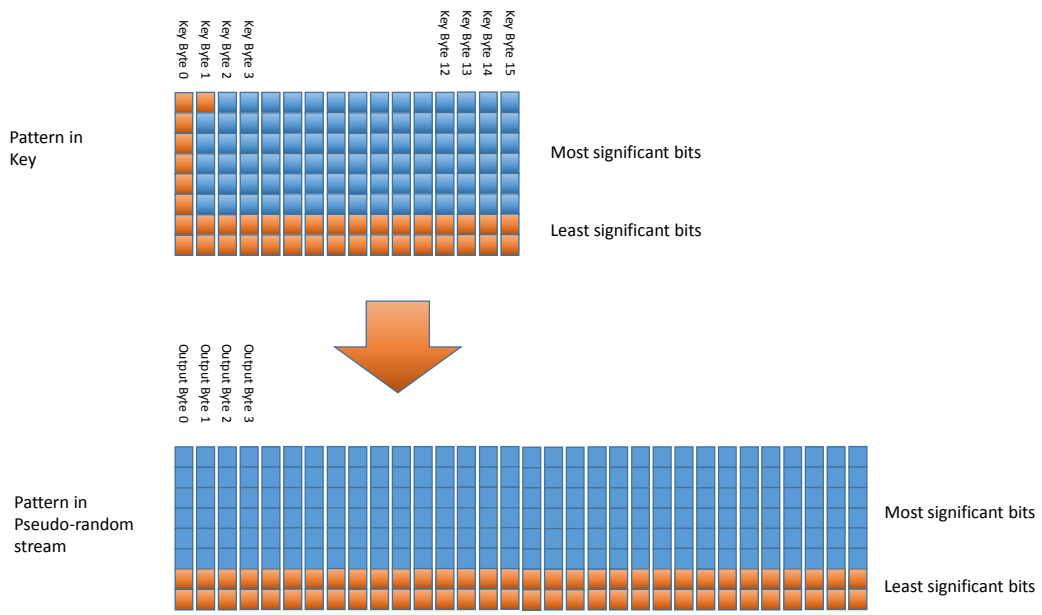
RC4 is not a secure cipher.
Back in the 90s, when RC4 was a trade secret of RSA and its details where known but not formally approved, RC4 was believed to be secure. However, in the following decade, RC4 had undergone significant scrutiny by cryptography experts, which showed statistical biases in the pseudo-random stream that allow an attacker to distinguish RC4 from random ([9], [10]) and to predict its allegedly pseudo-random bits with high probability ([10]).
While this statistical analysis requires many millions of RC4 keystream bytes, two researches in 2001 on the initialization of RC4 had switched the focus of RC4 analysis to its poor initialization mechanisms. The first [11] had shown that the second byte of RC4 has huge statistical bias, having twice the expected probability to be zero. The second [12], later known as the FMS research, had completely broken RC4 in the way it was used in the WEP protocol. These results had initiated a wave of research efforts and analysis results on RC4 initialization, a wave whose end is yet to be seen. Correlation between key bytes and state bytes (e.g., [13]), correlation between key bytes and first output bytes (e.g., [12]), and between key bytes and further output bytes ([14]), significant statistical biases in the first 256 output bytes [17], and many other results.

Interestingly, the series of successful attacks didn't have a significant impact on the popularity of RC4, and in fact during the wave of CBC attacks on SSL in 2012, some experts recommended switching to RC4.

The Invariance Weakness

The FMS research [12] details two significant RC4 weaknesses. The IV weakness, resulting in a practical key recovery attack and complete break of RC4 in the WEP protocol, received most of the attention, leaving the other weakness - denoted in the paper as the ***Invariance Weakness*** - in the shadows for 13 years.

The ***Invariance Weakness***, is an L-shape key pattern in RC4 keys, which once it exists in an RC4 key, preserves part of the state permutation intact throughout the initialization process. This intact part includes the least significant bits of the permutation, when processed by the PRGA algorithm, determines the least significant bits of the allegedly pseudo-random output stream along a long prefix of the stream. These patterns, and a detailed explanation of how they are preserved, are described in detail in [12] and [13]. These biased stream bytes are XOR-ed with the plaintext bytes, resulting in significant leakage of plaintext bytes from the ciphertext bytes.



These patterns occur for different number of LSBs, a single LSB, 2 LSBs, 3 LSBs to 7 LSBs, resulting with different classes of weak RC4 keys. Due to the structure of these classes, each class contains the succeeding classes and thus the first class is the largest, denoted below as ***the Main Class***.

The portion of q-class for L-byte keys (which is the probability of a random key to be in the class) is $2^{-(qL+(9-q))}$. For 16-byte key the portion of the ***Main Class*** (1-class) is 2^{-24} (1 in 16 million) and the portion of 2-class is 2^{-39} (very rare). These numbers are shown in the following table.

# LSBs	Applicability	Class Probability (8-byte key)	Class Probability (16-byte key)
1	Keys with even number of bytes	2^{-16}	2^{-24}
2	Keys with number of bytes that is a multiple of 4	2^{-23}	2^{-39}

3	Keys with number of bytes that is a multiple of 8	2^{-30}	2^{-54}
4	Keys with number of bytes that is a multiple of 16	2^{-37}	2^{-69}

When a key from a ***q***-class is used, the following things happen:

- The initialization phase of RC4 fails to mix the state with key material properly, and preserves the K least significant bits of its internal state
- As a result, the initial state of RC4 has fixed non-mixed ***q*** LSBs
- ***q*** least significant bits of the first 30-50 bytes stream bytes comply with a deterministic pattern with significant probability
- ***q*** least significant bits of the first 30-50 plaintext bytes are exposed with significant probability

The probability of the ***q*** LSBs to comply with the pattern drops with the stream. This probability is demonstrated in the following diagrams for a single LSB, 2 LSBs and 3 LSBs.

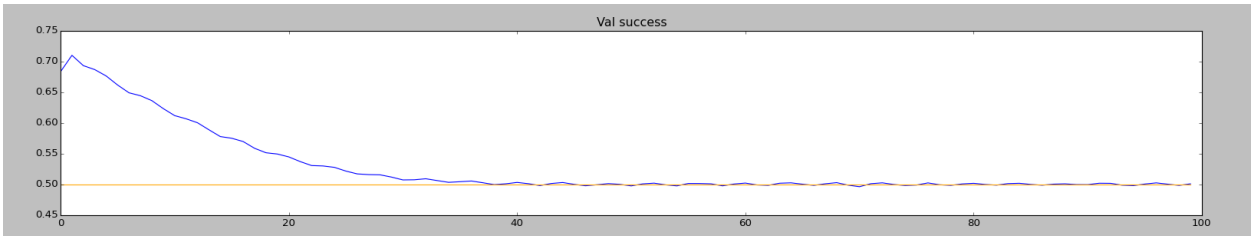


Figure 1: Single LSB (advantage over 0.5)

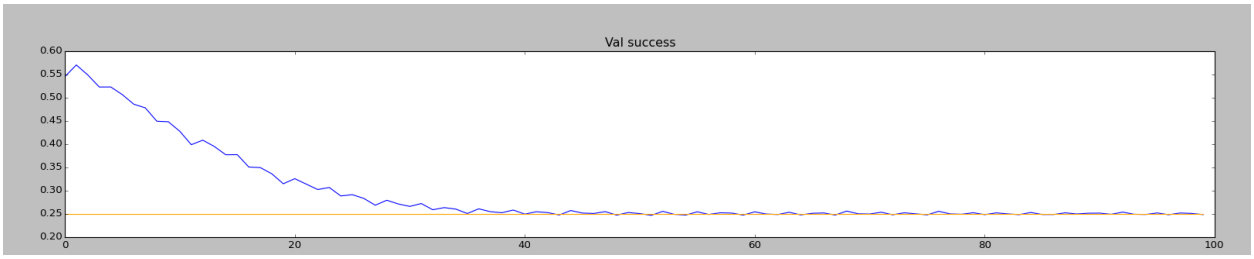


Figure 2: 2 LSBs (advantage over 0.25)

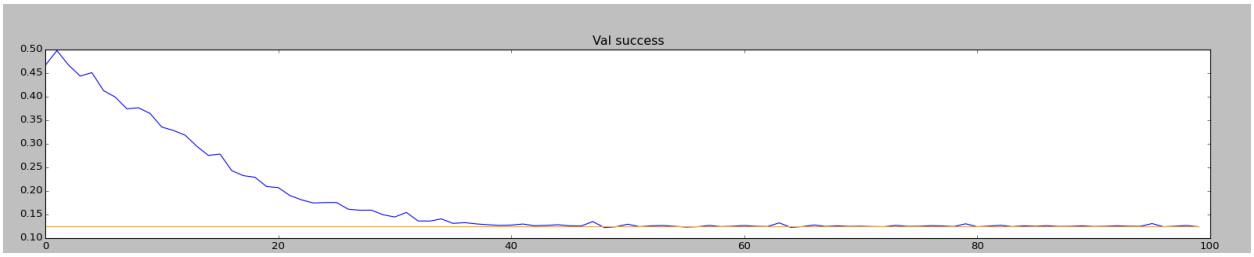


Figure 3: 3 LSBs (advantage over 0.125)

The reason for the decreasing probability is the state pattern getting “ruined” with the stream generation and after 50 bytes emitted by RC4 PRGA, the pattern fades out.

However, subsequent analysis we ran on RC4 streams when using weak keys showed a difference pattern between stream LSBs, which is less sensitive to this “ruining” effect, and manages to survive for as many as 100 bytes of the keystream.

The survival probability of difference patterns is demonstrated in the following diagrams for a single LSB, 2 LSBs and 3 LSBs, with reference to the value patterns (the diff pattern is in red and the value pattern remains in blue).

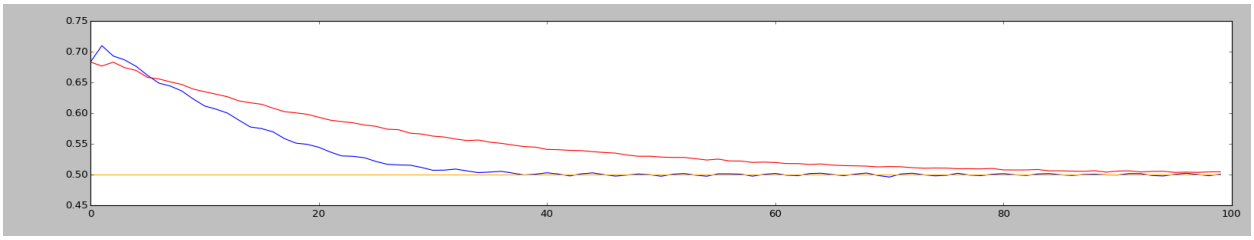


Figure 4: Single LSB (diff pattern; advantage over 0.5)

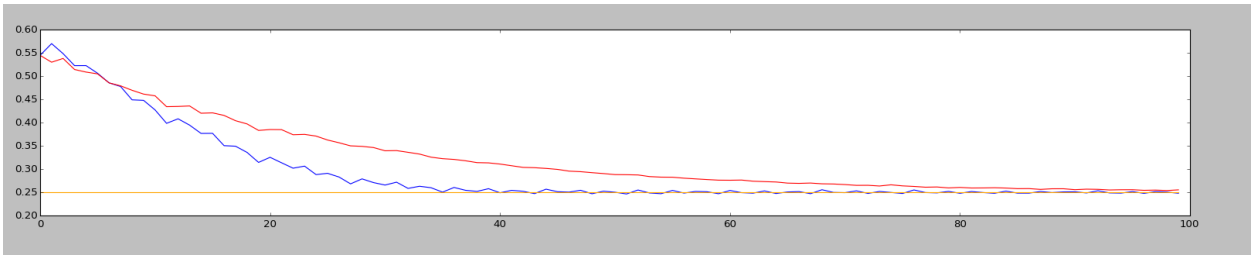


Figure 5: 2 LSBs (diff pattern; advantage over 0.25)

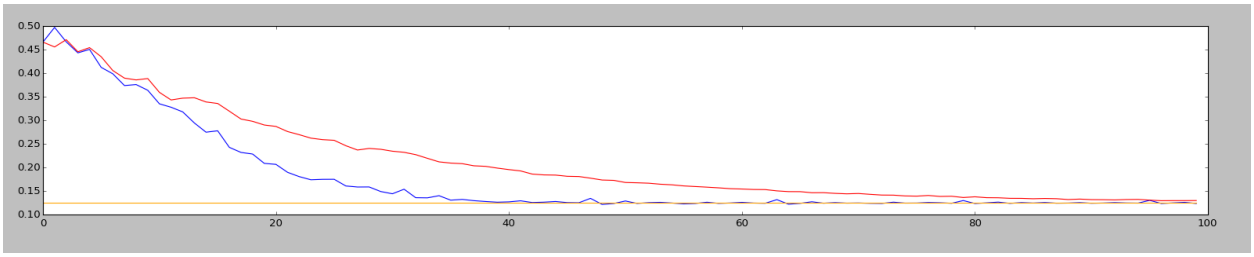


Figure 6: 3 LSBs (diff pattern; advantage over 0.125)

Applications of the Invariance Weakness

The Invariance Weakness of RC4 has several cryptanalytic applications, described in detail in [12] and [13], including statistical biases in the RC4 pseudo-random stream that allow an attacker to distinguish RC4 streams from randomness and enhancement of tradeoff attacks on RC4. Another application of the Invariance Weakness, which we use for our attack, is the leakage of plaintext data into the ciphertext when q-class keys are used.

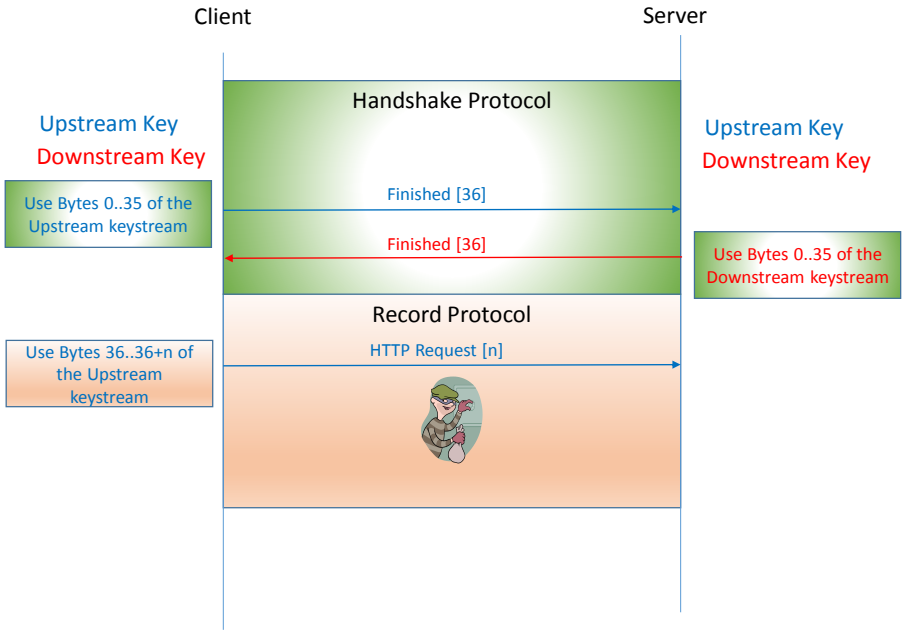
The authors of [8] had translated statistical biases in the keystream into plaintext leakage attacks. We follow [8] and use our statistical bias to recover plaintext information. ***The Invariance Weakness*** biases are not as strong as the biases used in [8]. However, these biases have unique characteristics, on one hand occurring rarely, but on the other hand effective in 100 keystream bytes with extremely high probability, opening the door to plaintext leakage attacks in several circumstances that were believed to be completely secure.

Using the Invariance Weakness to Attack SSL

SSL Usage of RC4

SSL Record Protocol uses RC4 for encryption in many SSL cipher suites. In the Handshaking protocol, RC4 encryption keys are generated for upstream and downstream communication. In the Record protocol, the upstream key is used for encryption of client-to-server communication, whereas the downstream key is used for encryption of server-to-client communication. It is important to note that the encryptions are statefull, using the first keystream bytes for encrypting the first message, the succeeding keystream bytes for encrypting the next message, etc. Given that the Invariance weakness is expressed only in the first 100 bytes of the keystream, it can be used only for the first 100 bytes of the protected upstream traffic and the first 100 bytes of the protected downstream traffic. Given that the first encrypted message in each direction is the SSL Handshake ***Finished*** message (36-bytes in typical usage of SSL), about 64 bytes of secret plaintext data are left for the attack.

This flow is depicted in the following diagram.



The first 36 bytes of the upstream keystream are used for encrypting the ***Finished*** message. The next bytes are used to encrypt the actual application data.

The Attack Scenario

Our attacks are based on the following scenario: the attacker sniffs a large number of SSL connections encrypted with RC4, waiting for a "hit"; that is the arrival of a weak key. Once a weak key arrives, the attacker predicts the LSBs of the keystream bytes, and uses these to extract the LSBs of the plaintext bytes from the ciphertext with significant advantage.

In order to fulfill this scenario, the attacker needs to determine which SSL sessions are the ones in which weak keys were used. For this isolation the attacker can use the fact that the first encrypted bytes include the SSL "***Finished***" message and HTTP request, both having predictable information. Thus, when a weak key is used, the plaintext patterns are XOR-ed with keystream patterns, generating ciphertext patterns visible to the attacker.

[7], [8] and other previous attacks on SSL, use tiny statistical biases to aggregate tiny pieces of plaintext information. In order to make this aggregation possible, the target object must be encrypted many times, with the same key in [7 and with different keys in a broadcast scenario in [8]. As opposed to these attacks, our attack scenario, when a weak key arrives, gets at once a significant amount of data on the target object, providing immediate partial plaintext recovery. On the other hand, this partial plaintext recovery cannot be extended into full plaintext recovery by continuing the attack and listening to more sessions.

What can you do with LSBs?

The above scenario allows the attacker to recover the least significant bits of up to a hundred secret bytes. When the target object is a session cookie, the attacker reduces the effective size of the cookie, allowing acceleration of brute force attack on the session cookie for the sake of Session Hijacking. For

example, learning the LSBs of ASP session cookies, which contain 16 characters of 5-bit entropy each, reduces 16 bits of the cookie entropy and thus can be used for faster brute force attack on the session cookie value. For PHP session cookies this acceleration can increase to up to a factor of 2^{32}.

When the target object is an n-character password, an attacker possessing the password LSBs can accelerate a dictionary attack with a factor of 2^n, reducing the security of an 8-char password by a factor of 256. Furthermore, an attacker who runs a brute force attack on a login API, can scan a database of known popular passwords, extract the passwords that comply with the LSB pattern and try only these, reducing the number of attempts by a factor of 2^n.

We ran an experiment on a database of most commonly used passwords, where we group passwords according to their least significant bits. For a single LSB the most common 1000 passwords were grouped into 252 sets, leaving the brute force attacker an average of only 4 attempts, below the barrier for most brute force protection policies. The estimation for the number of web accounts that are protected with one of the top 1000 most commonly-used passwords varies between 10-15 percent, setting the stage for an attack that uses the Invariance Weakness to extract the LSBs of the password in hope that the user is one of the careless 10%.

We used a passwords popularity analysis from [18] to estimate the number of brute force attempts an attacker needs to make once he has the LSBs of a password known to be in the top 100, 1000 and 10,000 (we took the numbers from [18] as is).

	Portion of web accounts	Number of LSB groups	Brute force worst case (#Attempts)	Brute force average case (#Attempts)
Top 100	4.4%	68	6	1.5
Top 1000	13.2%	252	24	4
Top 10,000	30%	557	201	18

The entropy of a 16-digit credit card number is usually considered to be 5 digits, since the first 6 digits are non-secret, the last 4 digits are freely exposed on receipts and for validation purposes, and there is a 1-byte checksum (Luhn algorithm). An attacker possessing the LSBs of a credit card number, reduces the search domain from 100,000 into only 1500. These 1500 candidate numbers can be tested by making attempts for small amount payments in retail web applications to find the valid one after 750 attempts on average. Thus the reduction in the credit card entropy from 100,000 potential numbers into 1500 potential numbers is significant, and increases the practicality of attacks.

A Man-in-the-Middle Attack

The first attack we describe resembles the RC4 attack from [8], with the attacker using a large number of encryptions of secret data, e.g., a session cookie, in order to recover parts of this cookie. One way to obtain this large number of encryptions is through the BEAST scenario, where JavaScript malware is downloaded from an attacker-controlled website and runs in the victim's browser, repeatedly sending HTTPS requests to the victim web server. Session cookies are automatically included in each of these requests in a predictable location, and can thus be targeted in the attack. The attacker, who needs new SSL connections for new RC4 keystream prefixes, can enforce termination of the SSL session after the

target encrypted cookie is sent; the browser will automatically establish a new SSL session when the next HTTPS request is sent.

The attack in [8] recovers the session cookie with probability of 50% after 2^{26} sessions. Our attack is expected to have a hit – 1-class key being used – every 2^{24} connections. Since such a hit is translated to long keystream pattern with probability of 1%-5%, several dozen hits are required to complete a successful partial plaintext recovery. For the sake of simplicity, in the rest of the discussion we use a single number of 1 billion as the number of attempts required to mount the attack, reflecting the number of encryptions the attacker is expected to see until being able to recover the data.

As opposed to the BEAST attack, the current attack does not require that the same key is used in all encryptions, but follows [8] in requiring that the key be changed between encryptions.

Moreover, as opposed to the attack in [8], which is highly sensitive to situations wherein the session cookie expires or the browser is closed by the user (in which case the attack has to begin from the start), our attack is completely insensitive to these cases. When a weak key is used, the learnt information is on the session cookie from that particular request, regardless of anything that happened before the hit, and anything that will happen after the hit.

A Non-Targeted Passive Attack

The nature of the ***Invariance Weakness***, allowing the attacker to learn significant plaintext data from a single hit (that is a single message that was encrypted with a weak key) opens the door to new attack scenarios which were not possible in any of the previous attacks. The next attack we'll describe is a passive variant of the aforementioned attack.

In it, the attacker eavesdrops on the inbound traffic to a popular retail or financial web application, aiming to steal credit card number info (a similar attack works on passwords). The attacker is required to wait 1 billion connections for a weak key usage event, this event being visible through fixed or structured data in this application, propagating through the keystream patterns into the ciphertext. The attacker then uses the Invariance Weakness to predict keystream LSBs, and uses these to calculate plaintext LSBs from ciphertext LSBs. It is important to note that the compromised credit card number or password is of a random victim, with the attacker not having any control over its identity.

According to Facebook user statistics, the number of daily active users in Facebook is close to 1 billion. The number of times each of these users executes a login, either visiting Facebook several times, or pressing a Like button in another application, can be roughly estimated as 4. Thus a passive eavesdropper sniffing on Facebook inbound lines waiting for hits, will see 256 weak keys on the daily 4 billion logins, 4 of which are expected to generate a long stream pattern and expose the password LSBs. Luckily Facebook had recently removed RC4 from its list of SSL supported ciphers.

Group Attacks

In the BEAST-like variant the attacker is required to generate 1 billion connections from the victim's browser. On the other hand, in the passive variant the attacker needs to sniff on 1 billion connections to the same web application. In another variant of the attack, combining the two, the attacker obtains these 1 billion connections actively from a group of victim users. The attacker needs to get Group Man-in-the-Middle setup, being a man in the middle for a group of users. Since the most natural methods of establishing man-in-the-middle are DNS poisoning and making users connect to a malicious hotspot

(either of which will establish the attacker as Man-in-the-Middle for a group of users), this Group Man-in-the-Middle situation is practical.

The attacker then runs the BEAST-like attack with each of the potential victims, terminating their connections immediately after the session cookie being sent, until one of them gets a successful hit, a weak key that propagates successfully into a long pattern. This event is expected to occur after a total number of 1 billion SSL connections from all users altogether. This attack scenario is unique, and stems from the unique nature of this attack, with every hit leaking a large quantity of secret information.

One-Time Encryption

From the perspective of a victim, the severity of an attack is proportional to the damage of the attack and to its likelihood. Consider a user who uses SSL to protect his most precious secret, and sends it only once over SSL. Interestingly, this one-time operation has a risk of one in 64 million to get a weak RC4 key and one in a billion to leak parts of his precious secret. It is true that one in a billion is a tiny fraction, but still, it is not a negligible fraction. The implication is the disturbing fact that every web user is clearly risking his data every single time he sends it over an RC4/SSL connection.

Conclusion and Recommendations

In this paper we demonstrated how the Invariance Weakness can be used to mount new attacks on SSL when using RC4. We improved the size of the prefix for which plaintext info leaks into 100 bytes. We showed how the “Reset Insensitivity” of the Invariance Weakness sets the stage for new attack scenarios, including the first passive attack on SSL.

The security of RC4 has been questionable for many years, in particular its initialization mechanisms. However, only in recent years has this understanding begun translating into a call to retire RC4. In this research we follow [8] and show that the impact of the many known vulnerabilities on systems using RC4 is clearly underestimated.

While waiting for a broad-brush retirement of RC4, specific parties should consider the following actions to protect themselves from its weaknesses:

- Web application administrators should strongly consider disabling RC4 it in their applications’ TLS configurations.
- Web users (particularly power users) are encouraged to disable RC4 it in their browser’s TLS configuration.
- Browser vendors would do well to consider removing RC4 from their TLS cipher lists.
- Organizations leveraging Imperva SecureSphere to protect their business-critical web applications and data, and wherein SecureSphere is set to handle TLS connections on behalf of the applications, can configure SecureSphere to stop using the weak ciphers and work only with robust ciphers.

References

1. T. Dierks and C. Allen. The TLS Protocol Version 1.0. RFC 2246, Internet Engineering Task Force, Jan. 1999. URL http://www.rfc-editor.org/rfc/rfc2246.txt.
2. T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.1. RFC 4346, Internet Engineering Task Force, Apr. 2006. URL http://www.rfc-editor.org/rfc/rfc4346.txt.
3. T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246, Internet Engineering Task Force, Aug. 2008. URL http://www.rfc-editor.org/rfc/rfc5246.txt.
4. T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3, http://tools.ietf.org/html/draft-ietf-tls-tls13-04
5. B. Canvel, A. Hiltgen, S. Vaudenay, and M. Vuagnoux. Password interception in a SSL/TLS channel. Advances in Cryptology-CRYPTO 2003, pages 583–599, 2003.
6. T. Duong and J. Rizzo. Here come the ⊕ Ninjas. 2011. http://www.hit.bme.hu/~buttyan/courses/EIT-SEC/abib/04-TLS/BEAST.pdf
7. N. AlFardan and K. G. Paterson. Lucky 13: Breaking the TLS and DTLS record protocols. In IEEE Symposium on Security and Privacy, 2013. URL http://www.isg.rhul.ac.uk/tls/Lucky13.html.
8. Nadhem J. AlFardan, Daniel J. Bernstein, Kenneth G. Paterson, Bertram Poettering, Jacob C. N. Schuldt. On the Security of RC4 in TLS and WPA, USENIX Security Symposium 2013
9. S. R. Fluhrer and D. McGrew. Statistical analysis of the alleged RC4 keystream generator. In B. Schneier, editor, FSE, volume 1978 of Lecture Notes in Computer Science, pages 19–30. Springer, 2000.
10. I. Mantin. Predicting and distinguishing attacks on RC4 keystream generator. In R. Cramer, editor, EUROCRYPT, volume 3494 of Lecture Notes in Computer Science, pages 491–506. Springer, 2005.
11. I. Mantin and A. Shamir. A practical attack on broadcast RC4. In M. Matsui, editor, FSE, volume 2355 of Lecture Notes in Computer Science, pages 152–164. Springer, 2001.
12. S. R. Fluhrer, I. Mantin, and A. Shamir. Weaknesses in the key scheduling algorithm of RC4. In S. Vaudenay and A. M. Youssef, editors, Selected Areas in Cryptography, volume 2259 of Lecture Notes in Computer Science, pages 1–24. Springer, 2001.
13. I. Mantin. Analysis of the stream cipher RC4. Master Thesis, the Weizmann Institute of Science.
14. I. Mantin. A Practical Attack on the Fixed RC4 in the WEP Mode. In Advances in Cryptology - ASIACRYPT 2005
15. This POODLE Bites: Exploiting the SSL 3.0 Fallback. Google Security Advisory https://www.openssl.org/~bodo/ssl-poodle.pdf
16. A Perfect CRIME? Only TIME Will Only TIME Will Tell. Tal Be'ery, Amichai Shulman. https://media.blackhat.com/eu-13/briefings/Beery/bh-eu-13-a-perfect-crime-beery-wp.pdf
17. S. Sen Gupta, S. Maitra, G. Paul, and S. Sarkar. (Non-) random sequences from (non-) random permutations – analysis of RC4 stream cipher. Journal of Cryptology, pages 1–42, 2012.
18. 10,000 Top Passwords. https://xato.net/passwords/more-top-worst-passwords/#.VPiyH_ysVew