Sweet32: Birthday attacks on 64-bit block ciphers in TLS and OpenVPN

CVE-2016-2183, CVE-2016-6329

Cryptographic protocols like TLS, SSH, IPsec, and OpenVPN commonly use block cipher algorithms, such as AES, Triple-DES, and Blowfish, to encrypt data between clients and servers. To use such algorithms, the data is broken into fixed-length chunks, called blocks, and each block is encrypted separately according to a mode of operation. Older block ciphers, such as Triple-DES and Blowfish use a block size of 64 bits, whereas AES uses a block size of 128 bits.

It is well-known in the cryptographic community that a short block size makes a block cipher vulnerable to birthday attacks, even if there are no cryptographic attacks against the block cipher itself. We observe that such attacks have now become practical for the common usage of 64-bit block ciphers in popular protocols like TLS and OpenVPN. Still, such ciphers are widely enabled on the Internet. Blowfish is currently the default cipher in OpenVPN, and Triple-DES is supported by nearly all HTTPS web servers, and currently used for roughly 1-2% of HTTPS connections between mainstream browsers and web servers.

We show that a network attacker who can monitor a long-lived Triple-DES HTTPS connection between a web browser and a website can recover secure HTTP cookies by capturing around 785 GB of traffic. In our proof-of-concept demo, this attack currently takes less than two days, using malicious Javascript to generate traffic. Keeping a web connection alive for two days may not seem very practical, but it worked easily in the lab. In terms of computational complexity, this attack is comparable to the recent attacks on RC4. We also demonstrate a similar attack on VPNs that use 64-bit ciphers, such as OpenVPN, where long-lived Blowfish connections are the norm.

Countermeasures are currently being implemented by browser vendors, OpenSSL, and the OpenVPN team, and we advise users to update to the latest available versions.

Our results are described in the following technical paper, presented at ACM CCS 2016:

On the Practical (In-)Security of 64-bit Block Ciphers — Collision Attacks on HTTP over TLS and OpenVPN

Karthikeyan Bhargavan, Gaëtan Leurent

Block Ciphers and the Birthday Bound

The security of a block cipher is often reduced to the key size k: the best attack should be the exhaustive search of the key, with complexity 2^k. However, the block size n is also an important security parameter, defining the amount of data that can be encrypted under the same key. This is particularly important when using common modes of operation: we require block ciphers to be secure with up to 2^n queries, but most modes of operation (e.g. CBC, CTR, GCM, OCB, etc.) are unsafe with more than $2^{n/2}$ blocks of message (the birthday bound).

With a modern block cipher with 128-bit blocks such as AES, the birthday bound corresponds to 256 EB. However, for a block cipher with 64-bit blocks, the birthday bound corresponds to only 32 GB, which is easily reached in practice. When the amount of data encrypted under a fixed key approaches this limit, the security guarantees of the mode of operation start to crumble. This problem is well-known by cryptographers, who always require keys to be changed **well before** $2^{n/2}$ blocks. However it is often minimized by practitioners

because the attacks require known plaintext, and reveal only little information. Indeed, standard bodies only recommend to change the key just before $2^{n/2}$ blocks, and many implementations don't enforce any limit on the use of a key.

In particular, there are many uses of block ciphers with 64-bit blocks where large amount of data are potentially encrypted under the same key, such as:

- 3G telephony (UMTS), encrypted with KASUMI;
- OpenVPN, which uses Blowfish as the default cipher;
- many Internet protocols, such as TLS, IPSec and SSH, support Triple-DES as a legacy cipher.

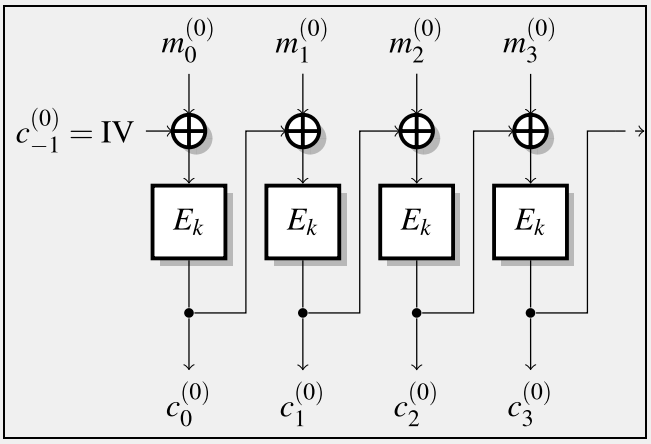
In all these scenarios, 32 GB of data can be transferred in less than one hour with a fast connection.

Exploiting Block Cipher Collisions

In practice, block ciphers are used with a mode of operation in order to deal with messages of arbitrary length. The CBC mode is one of the oldest encryption modes, and still widely used. The message M is divided into blocks m_i and is encrypted as:

$$c_i = E_k(m_i \oplus c_{i-1}),$$

where c_{-1} is an initialization value usually denoted as IV. We now explain the impact of collisions on the CBC mode.



The CBC mode

CBC has been proven secure up to $2^{n/2}$ of messages. On the other hand there is a simple birthday attack against CBC: after $2^{n/2}$ message blocks encrypted with the same key (in the same message or in different messages), a collision between two ciphertext blocks $c_i = c_j$ is expected. Since E_k is a permutation, a collision in the output means that the inputs are the same ($m_i \oplus c_{i-1} = m_j \oplus c_{j-1}$) which reveals the xor of two plaintext blocks:

$$m_i \oplus m_j = c_{i-1} \oplus c_{j-1}.$$

With 2^d blocks of data the expected number of collisions is roughly 2^{2d-n-1} (following the birthday paradox).

Practical impact

In many contexts, recovering only the xor between two plaintext blocks is not sufficient for an attack with a practical impact. However, an attack can be mounted when the following conditions are fulfilled:

- a fixed secret is sent repeatedly;
- some fraction of the plaintext is known.

In this case, there is a chance that a collision leaks the xor between the fixed secret and the known plaintext; this would immediately reveal the secret. More precisely, this attack is expected to succeed with high probability as soon as 2^s copies of the secret and 2^t known blocks are encrypted, with $s+t \geq n$.

In particular, this is applicable to the security of encrypted HTTP connections, when an authentication token is send with every request. A man-in-the-browser attacker can generate a large number of requests, and recover

the authentication token, using the fact that most headers in the request are predictable or even controlled by the attacker.

64-bit Block Cipher Usage on the Internet

Many of the most influential Internet security protocols, such as TLS, SSH, and IPsec were standardized at a time when 64-bit block ciphers, such as Triple-DES and Blowfish, were still considered strong. For example, in TLS 1.0 and 1.1, Triple-DES is the mandatory encryption algorithm, so all TLS libraries implement it and a vast majority of web servers support it. In addition, until we disclosed the attacks in this paper, OpenSSL included Triple-DES ciphersuites in its HIGH-security list (it has now been moved to MEDIUM).

Triple-DES and Blowfish usage in VPNs.

IPSec. Most IPSec-based VPN clients support Triple-DES for interoperability. In particular, some versions of Microsoft's L2TP VPN client use Triple-DES by default.

OpenVPN. OpenVPN is a popular open-source VPN solution originally written by James Yonan. The **default encryption** for the transport protocol of OpenVPN **is Blowfish – a 64-bit cipher –** with the CBC mode. OpenVPN supports two different modes to generate session keys to encrypt the messages:

- In **pre-shared-key mode**, static keys are used for all the traffic. In particular, there is no limit to the lifetime of those keys.
- In **TLS mode**, session keys are generated with a TLS handshake, using certificates to authenticate the peers. The session keys are updated periodically, with limits on the number of packets, the number of bytes, or a session time. The default configuration rekeys the tunnel every hour.

Triple-DES usage in HTTPS.

Triple-DES is the second most widely supported cipher (after AES) in HTTPS servers, with about 87% of servers supporting it. Furthermore, all popular web browsers support Triple-DES. The cipher that is actually negotiated for a TLS connection is chosen by the server, based on its local preference order and on the order in which the client advertises its ciphersuites. Since most modern browsers and servers prefer AES over Triple-DES one may expect only a negligible number of connections to negotiate Triple-DES. However, we find evidence that 1-2% of all TLS connections likely use Triple-DES in CBC mode, as described below.

Firefox Telemetry. Telemetry data from Mozilla Firefox show that Triple-DES is used for close to 1% of HTTPS connections from Firefox browsers (0.76% with beta 49). The use of Triple-DES with Firefox is slowly decreasing, and peaked with the removal of RC4 from the list of supported ciphers in version 36. Indeed a number of servers are configured to use, in order of preference, first RC4, then Triple-DES, and now use Triple-DES with Firefox. Since all modern browsers have deprecated RC4 between 2013 and 2015 (following RFC 7465), they will also use a Triple-DES ciphersuite in this situation.

Scanning the Top 1M Websites. We performed a scan of the top 1 million servers as listed by Alexa using the cipherscan tool. We found that 86% of the servers that support TLS include Triple-DES as one of the supported ciphers. Moreover, 1.2% of these servers are configured in such a way that they will actually pick a Triple-DES based ciphersuite with a modern browser, even though better alternatives are available. (In particular many of these servers support AES-based ciphersuites, but use Triple-DES or RC4 preferentially.)

Windows XP Clients and Windows 2003 Servers. The Windows Server 2003 operating system does not support AES-based ciphersuites in its default configuration, although support for AES can be added with an optional hotfix. The Windows XP operating system with security update MS10-049 supports AES-based ciphersuites. If AES-based ciphersuites have not been added, these operating systems support only RC4, Triple-DES, DES, and RC2-40. While they are not supported anymore by Microsoft, they still have some users, and this creates situations where the best available cipher is Triple-DES.

Long-lived HTTPS connection

An important requirement for the attack is to send a large number of requests in the same TLS connection. Therefore, we need to find client and servers that not only negotiate the use of Triple-DES, but also exchange a large number of HTTP request in the same TLS connection (without rekeying). This is possible using a persistent HTTP connection, as defined in HTTP/1.1 (Keep-Alive). On the client side, all browsers that we tested (Firefox, Chrome, Opera) will reuse a TLS connection as long as the server keeps it open.

On the server side, we found that a number of HTTP servers will close the TLS connection even when it is still active. In particular, Apache and Nginx limit the number of requests sent in the same connection, with a maximum of 100 in the default configuration. On the other hand, IIS does not seem to have such a limit. In practice, many high profile servers accept a very large number of requests in a single TLS connection.

Vulnerable Websites. For a better estimate of the number of vulnerable servers, we tested servers from Alexa's top 10k that negotiate Triple-DES with a modern client. We identified 11483 different HTTPS servers, and found that 226 of them (1.9%) negotiate Triple-DES with a modern client. Moreover, 72 of these (0.6% of the total) also accept to keep a connection open for at least 800k requests. Consequently, the duration of the attack is not unrealistic, at least from the viewpoint of browsers and servers, and we estimate that at least 0.6% of HTTPS connections are vulnerable to our attacks.

For advice about how to check a website's configuration and configure it properly, see the FAQ.

Attacking Authenticated HTTP over TLS and OpenVPN

We now demonstrate concrete attacks against authenticated HTTP sessions even when they are secured by TLS or OpenVPN. First, we identify a few examples of secret authentication credentials that are repeatedly sent by the browser on every request. We then show how we can recover these secrets using block cipher collisions.

HTTP Bearer Tokens

Cookie-based Sessions. Modern HTTPS websites use a variety of methods to manage authenticated sessions with their clients. The most popular mechanism is secure *cookies* as specified in RFC6265. Once a user has logged in, the server sets a cookie containing a secret value on the user's browser. The browser will then send the cookie on all subsequent requests to the website, implicitly authenticating the user.

Cookies are sensitive, because an attacker who obtains a session cookie can then log in as the user from a different browser. The cookie acts as a *bearer token* that carries the user's delegated credential.

A cookie for a website is normally included in all requests to that website whether that request was initiated by the user, or a script on the website, or even by a different website. Browsers sometimes impose stricter rules for this last category of requests, called cross-origin or cross-domain requests. For example, XmlHttpRequests sent from one domain to another may not have cookies attached. However, cross-domain requests for images or iframes will still send cookies.

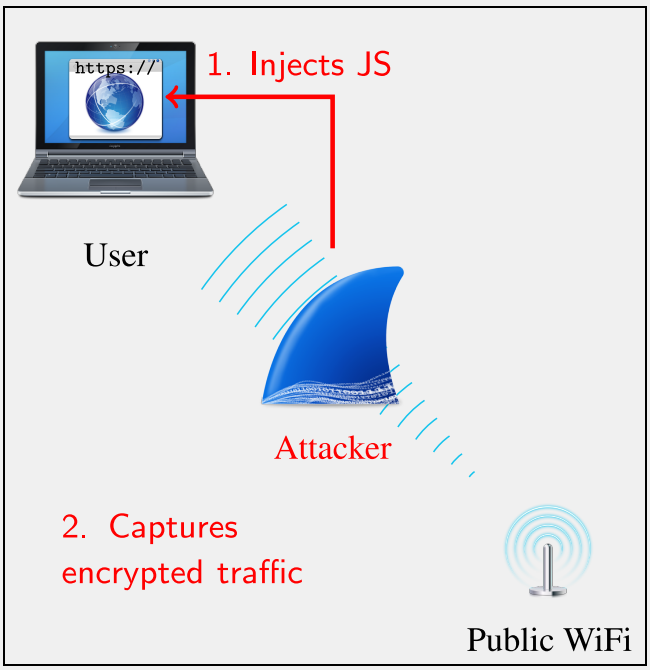
HTTP BasicAuth. Apart from cookies, there are other mechanisms for a website to authenticate the user. In the HTTP Basic Authentication mechanism specified in RFC7617, the browser asks the user to enter a username and password into a special dialog and then sends this information (in plaintext) as an HTTP header of the form:

```
Authorization: Basic dGVzdDoxMjPCow=
```

Once a user has entered his login information once, the browser will typically cache this information and use it on all subsequent requests to the server. Notably, even if a different website creates a cross-domain request to the authenticated server, the BasicAuth credentials will be automatically sent by the browser. BasicAuth

credentials contain the user's password and are hence security-critical. One should only use this authentication mechanism over HTTPS but a number of corporate websites use BasicAuth over HTTP, under the assumption that their users can only access the website over a VPN or some other secure connection.

The Beastly Attack Scenario



The Beastly attacker

Our attack scenario, is similar to the setup used in recent attacks on RC4. The attacker wants to steal some bearer token that is being repeatedly sent by a browser to a website secured with HTTPS, or an HTTP website accessed through a VPN. We assume that the attacker can control some JavaScript on a web page loaded by the user's browser, either by actively tampering with an HTTP response on the wire, or by hosting a malicious website that the user is fooled into visiting. We also assume that the attacker can observe all the encrypted traffic between the target browser and the secure website.

We further assume that the data is encrypted with a 64-bit block cipher in CBC mode (either an HTTPS connection where the client and server have negotiated Triple-DES, or an HTTP connection though a VPN encrypted with Blowfish or Triple-DES). Suppose the victim is already logged in to a website and has a session cookie. The attacker runs malicious JavaScript code on the victim's browser that repeatedly sends HTTP queries to the target website server, each containing the session cookie. If he sends close to 2^{32} queries, a collision is expected between a ciphertext block corresponding to the cookie (c_i), and a known block (c_j), containing a known part of the query. The collision attack against CBC reveals the session cookie:

$$p_i = p_j \oplus c_{i-1} \oplus c_{j-1}.$$

Note that most of the traffic generated by the attacker is known or predictable. The requested URL is chosen by the attacker, and all the headers excepted the cookie are predictable and can be observed in any HTTP traffic.

Proof-of-Concept Attack Demo

The attacker code consists of two parts: a JavaScript program that sends a large number of HTTP requests, and a network adversary who processes the resulting ciphertext to recovers a 16-byte secret.

Man-in-the-browser code. The attacker code is shown on the right: it uses Web workers to issue `XmlHttpRequests`, using the `withCredentials` property to make sure that cookies are send in the cross-origin request. We experimented with several browsers, and we obtained the best results with Firefox Developer Edition 47.0a2.

```
<html>
  <body>
    <script>
    var W = new Array;
    for (var i=0; i<8; i++) {
        var x = new Worker("worker.js");
        W.push(x);
    }
    </script>
  </body>
</html>
```

attack.html

Recovering collisions. We captured the encrypted packets with `tcpdump` and used a C++ program to

```
var url = "https://10.0.0.1/index.html";
var xhr = new XMLHttpRequest;

// Expand URL to ~4kB using a query string
// Alternatively, force a large cookie
url += "?";
```

extract the ciphertext blocks (using `libpcap`). In both the HTTPS attack and the OpenVPN attack, each HTTP query is sent in a separate encrypted record, which contain the plaintext at a fixed position, as well as some extra information (packet number, padding, MAC, ...). Therefore, it is easy to know to which plaintext block corresponds each ciphertext block, and to align the cookie to a block limit. After capturing all the traffic, the C++ program sorts the ciphertext blocks in order to locate collisions. Since the amount of data is quite large (hundreds of gigabytes), we use the external sort implementation of the `stxxl` library. With a NAS storage, sorting the data took around four hours.

```
var x = 10000000;
for (var i=0; i<=500; i++) {
    url += x++;
}


while(true) {
    xhr.open("HEAD", url, false);
    xhr.withCredentials = true;
    xhr.send();
    xhr.abort();
}
```

worker.js

Attacking Basic Auth over OpenVPN

To demonstrate the attack against OpenVPN, we use a pre-shared-key tunnel between two physical machines running Linux, with Firefox Developer Edition 47.0a2 on one side, and an nginx server on the other side. Access to the server is protected by BasicAuth, and the user has entered his credentials. Using the default OpenVPN settings, the tunnel is encrypted with Blowfish in CBC mode.

We use the Javascript code described in the previous section to send a large number of requests to the server though the tunnel. We found that increasing the size of the request to 4~kB does not significantly reduce the query rate, but reduces the number of queries required for the attack. In our setting, the browser generates about 2900 requests per second, using several web Worker running in parallel. We expect the first collision after roughly $2^{32.3}$ blocks (40 GB), *i.e.* one hour. In practice, we detected the first collision early, after only 30 minutes ($2^{31.3}$ blocks); as expected, the predicted xor difference was correct. The full attack needs about $2^{36.6}$ blocks (785 GB) to recover a two-block secret out of 4 kB messages; this should take about 19 hours in this setting. **In our demo, it took 18.6 hours and 705 GB, and we successfully recovered the 16-byte authentication token.**

Attacking Secure Cookies over HTTPS

The attack against HTTPS connection is very similar to the attack against OpenVPN, as long as we have a client and server that negotiate Triple-DES, and that keep a connection open for a long time.

Vulnerable Websites. As detailed above, about 0.6% of the top Alexa 100K websites support Triple-DES and allow long-lived HTTPS connections. This list contains many high-profile e-commerce and banking websites.

We use the Javascript code described above, but we note that with several `Workers` running in parallel, modern browsers open a few parallel connections to the server and split the requests over these connections. For our attack, we need to maximize the throughput over a *single* connection. This can be achieved by disturbing some of the connections so that most of request are send in a single. In our setup, we used `iptables` rules to limit the rate of all connections except one. In a real attack, this would be done by an active man-in-the-middle, but a passive man-in-the-middle can also mount the attack -- it will just take more time to collect the data.

Proof-of-Concept Attack Demo. In our setup we use Firefox Developer Edition 47.0a2 running on Linux and an IIS 6.0 server in a Windows Server 2003 R2 SP2 Virtual machine. The default configuration of IIS 6.0 with all recommended updates offers only RC4 and Triple-DES ciphers, and will use Triple-DES with Firefox and other recent browsers where RC4 is disabled. Moreover, IIS 6.0 supports HTTP/1.1 and keeps an active connection open for an arbitrary long time.

On Firefox Developer Edition 47.0a2, with a few dozen `Workers` running in parallel, we can send up to 2000 requests per second in a single TLS connection. In our experiment, we were lucky to detect the first collision

after only 25 minutes ($2^{20.1}$ requests), and we verified that the collision revealed the xor of two plaintexts blocks. As seen previously, the full attack should require $2^{36.6}$ blocks (785 GB) to recover a two-block cookie, which should take 38 hours in our setting. **Experimentally, we have recovered a two-block cookie from an HTTPS trace of only 610 GB, captured in 30.5 hours.**

Impact and Mitigation

We have demonstrated the first concrete attacks on mainstream Internet protocols that exploit block cipher collisions. Our attacks can recover valuable secrets such as HTTP cookies and passwords in under 40 hours. Our attacks impact a majority of OpenVPN connections and an estimated 0.6% of HTTPS connections to popular websites. We expect that our attacks also impact a number of SSH and IPsec connections, but we do not have concrete measurements for these protocols. Like many recent attacks on TLS, such a BEAST and RC4 NOMORE, the underlying principles behind our attacks were well known to cryptographers. Our goal is to raise awareness among practitioners about the vulnerabilities of short block ciphers and on safe ways of using them.

Comparison with RC4 attacks

Our attack scenario is very similar to the setup of the recent attacks on the use of RC4 in HTTPS. We use the same man-in-the-browser setting to generate a large number of HTTP requests, and the data complexity of the attack is comparable. Our attack requires only $2^{29.1}$ short queries of 512 bytes (280 GB in total), which can be reduced to $2^{27.6}$ longer queries of 4 kB (785 GB in total). However, these numbers are for the case when all the data is encrypted within the same session. Even if the amount of data sent on a single connection is limited, as long as the limit is close enough to the birthday bound, we can still mount our attacks across multiple parallel and sequential sessions, albeit with a higher data and time complexity.

Mitigation

The obvious way to avoid these attacks is to stop using legacy 64-bit block ciphers. Alternatively, the attack can be mitigated by rekeying the session frequently.

Concretely, we recommend the following measures to prevent our attack:

- Web servers and VPNs should be configured to prefer 128-bit ciphers. According to our scans, about 1.1% of the top 100k web server from Alexa, and 0.5% of the top 1 million, support AES but prefer to use 3DES.
- Web browsers should offer 3DES as a fallback-only cipher, to avoid using it with servers that support AES but prefer 3DES.
- TLS libraries and applications should limit the length of TLS sessions with a 64-bit cipher. This could be done with TLS renegotiation, or in some cases by closing the connection and starting a new one (i.e. limiting HTTP/1.1 Keep-Alive, SPDY, and HTTP/2 with 3DES ciphersuites).
- OpenVPN users can change the cipher from the default Blowfish to AES, using for instance `cipher AES-128-CBC` on the client and server configuration. If they don't control the server configuration, they can mitigate the attack by forcing frequent rekeying with `reneg-bytes 64000000`.

Responsible disclosure

We have communicated our results and concerns to the OpenVPN team, and to various website owners, browser vendors, and TLS libraries. They all acknowledged the issue, and are working on implementing countermeasures. The TLS vulnerability received CVE number CVE-2016-2183, and the OpenVPN vulnerability is tracked as CVE-2016-6329.

NIST is working on deprecation of 3DES. They plan to limit the use of 3DES to 2^{20} blocks with a given key, and to disallow 3DES in TLS, IPsec, and possibly other protocols.

OpenVPN 2.3.12 will display a warning to users who choose to use 64-bit ciphers and encourage them to transition to AES (cipher negotiation is also being implemented in the 2.4 branch). It will also implement a default renegotiation limit of 64MB when used in TLS mode in a future version. A wiki entry provides further details.

OpenSSL has moved 3DES ciphersuites from the HIGH category to MEDIUM in the 1.0.1 and 1.0.2 branches, and will disable it by default in the upcoming 1.1.0 release. They have a blog entry with further details.

Akamai will offer an option for web server administrators to drop 3DES from the offered ciphers.

Apple has disabled 3DES on icloud.com and is recommending that all its customers disable 3DES on their websites.

Currently, most browsers see about 1% of their connections using 3DES, and vendors consider this number too high to simply disable 3DES on the client side, since too many websites would be broken. So, they are instead considering implementing data limits per connection to force rekeying, or offering 3DES ciphersuites only in a fallback negotiation if no AES ciphersuite is acceptable to the server.

Mozilla is implementing data limits for all ciphersuites. This has been integrated into NSS 3.27, which should be used in Firefox 51.

Microsoft has removed 3DES from the False Start whitelist.

More details about implemented countermeasures will be added to this webpage as they become available.

On the Web

The attack has been mentionned on Threatpost, Ars Technica, Matthew Green's blog, Tom's hardware, Softpedia, Naked Security, The Register, inira.fr, and ZDnet.fr, among others... .

FAQ

Why is 3DES still used with a modern browser?

Because more than 1% of the web servers are poorly configured, and prefer using 3DES rather than AES.

How can I check if my server is well configured?

You can use the scanning tool form Qualys SSL Labs. In the "Handshake Simulation" section, you should see 3DES or RC4 only with browsers that don't support stronger ciphersuites, like IE6/XP and IE8/XP. If you have 3DES ciphersuites at the bottom of the "Cipher Suites" section, you can try to remove them, but it's not an immediate security issue. Removing 3DES will protect you against potential downgrade attack, but it will also break connections from older clients.

How can I fix my server's configuration

You can follow the advice from Mozilla, and their Configuration generator.

I'm using Blowfish-256, is it safe?

No, the attack is independent of the key length. It work on Blowfish with any key length, and Triple-DES with 3 independent keys (168 bits in total).

Why Sweet32?

It's a stupid pun, based on the sweet sixteen birthday celebration. Our attack is a birthday attack (taking its name from the birthday paradox) with complexity 2^{32}. You could also say that 2^{32} is the *sweet spot* where attacks become practical.

Can I use your logo to talk about the attack?

Yes, you can. To the extent possible under law, Gaëtan Leurent has waived all copyright and related or neighboring rights to the Sweet32 logo.

About us

We are a pair of researchers from INRIA, the French national research institute for computer science. You can contact us at our email addresses: [FirstName].[LastName]@inria.fr (use our names without any accents.)

- Karthikeyan Bhargavan
- Gaëtan Leurent