Return Of Bleichenbacher's Oracle Threat (ROBOT)

Hanno Böck
Juraj Somorovsky
Ruhr University Bochum, Hackmanit GmbH
Craig Young
Tripwire VERT

Abstract

In 1998 Bleichenbacher presented an adaptive chosen-ciphertext attack on the RSA PKCS #1 v1.5 padding scheme. The attack exploits the availability of a server which responds with different messages based on the ciphertext validity. This server is used as an oracle and allows the attacker to decrypt RSA ciphertexts. Given the importance of this attack, countermeasures were defined in TLS and other cryptographic standards using RSA PKCS #1 v1.5.

We perform the first large-scale evaluation of Bleichenbacher's RSA vulnerability. We show that this vulnerability is still very prevalent in the Internet and affected almost a third of the top 100 domains in the Alexa Top 1 Million list, including Facebook and Paypal.

We identified vulnerable products from nine different vendors and open source projects, among them F5, Citrix, Radware, Palo Alto Networks, IBM, and Cisco. These implementations provide novel side-channels for constructing Bleichenbacher oracles: TCP resets, TCP timeouts, or duplicated alert messages. In order to prove the importance of this attack, we have demonstrated practical exploitation by signing a message with the private key of `facebook.com`'s HTTPS certificate. Finally, we discuss countermeasures against Bleichenbacher attacks in TLS and recommend to deprecate the RSA encryption key exchange in TLS and the RSA PKCS #1 v1.5 standard.

1 Introduction

In 1998 Daniel Bleichenbacher published an adaptive chosen-ciphertext attack on RSA PKCS #1 v1.5 encryption as used in SSL [11]. In his attack the attacker uses a vulnerable server as an oracle and queries it with successively modified ciphertexts. The oracle answers each query with true or false according to the validity of the ciphertext. This allows the attacker to decrypt arbitrary ciphertext without access to the private key by using Bleichenbacher's algorithm for exploiting the PKCS #1 v1.5 format.

Instead of upgrading to RSA-OAEP [29], TLS designers decided to use RSA PKCS #1 v1.5 in further TLS versions and apply specific countermeasures [2, 17, 34]. These countermeasures prescribe that servers must always respond with generic alert messages. The intention is to prevent the attack by making it impossible to distinguish valid from invalid ciphertexts. Improper implementation of Bleichenbacher attack countermeasures can have severe consequences and can endanger further protocols or protocol versions. For example, Jager, Schwenk, and Somorovsky showed that the mere existence of a vulnerable implementation can be used cross-protocol to attack modern protocols like QUIC and TLS 1.3 that do not support RSA encryption based key exchanges [23]. Aviram et al. published DROWN, a protocol-level variant of Bleichenbacher's attack on SSLv2 [6].

Due to the high relevance of this attack, the evaluation of countermeasures applied in TLS libraries is of high importance. There were several researchers concentrating on the evaluation of Bleichenbacher attacks in the context of TLS. However, these evaluations mostly concentrated on the evaluation of the attacks in open source TLS implementations. Meyer et al. showed that some modern TLS stacks are vulnerable to variations of Bleichenbacher's attack [28]. For example, the Java TLS implementation was vulnerable due to handling of encoding errors and other implementations were demonstrated as vulnerable through time based oracles. In 2015 Somorovsky discovered that MatrixSSL was vulnerable as well [36].

While Bleichenbacher attacks have been found on multiple occasions and in many variations, we are not aware of any recent research trying to identify vulnerable TLS implementations in the wild. Given the fact that most of the open source implementations are secure

according to the latest evaluations [28, 36], one would think that such an evaluation would not reveal many new vulnerable implementations. But this is not the case. We developed a systematic scanning tool that allowed us to identify multiple vulnerable TLS hosts. Many of the findings are interesting from the research perspective since they uncover different server behaviors or show new side-channels which were specifically triggered by changing TLS protocol flows or observing TCP connection state. These behaviors are of particular importance for the analyses of different vulnerabilities relying on server responses, for example, padding oracle [37] or invalid curve attacks [24].

Contributions. Our work makes the following contributions:

- We performed the first large-scale analysis of Bleichenbacher's attack and identified vulnerabilities in high profile servers from F5, Citrix, Radware, Palo Alto Networks, IBM, and Cisco, as well as in the open source implementations Bouncy Castle, Erlang, and WolfSSL.
- We present new techniques to construct Bleichenbacher oracles which are of particular interest for developing related attacks. These involve changing TLS protocol flows or observing TCP connection states.
- We implemented a proof of concept attack that allowed us to sign a message with the private key of Facebook's web page certificate.
- Finally, we discuss the countermeasures proposed in TLS 1.2 [34] and whether it is feasible to deprecate RSA encryption based key exchanges.

Responsible disclosure and ethical considerations. In collaboration with affected web site owners we responsibly disclosed our findings to vulnerable vendors. We collaborated with them on mitigations and reevaluated the patches with our scripts. Several vendors and web site owners awarded us with bug bounties.

To raise the awareness of these attacks, we also collaborated with different TLS evaluation tool developers. The Bleichenbacher vulnerability check was afterwards included in SSL Labs and testssl.sh.

As a result of a successful attack, the attacker is able to obtain the decrypted RSA ciphertext or sign an arbitrary message with server's private key. Therefore, by performing our proof of concept attacks we were not able to reconstruct the RSA private key. We performed our attacks with dummy data and never attempted to decrypt real user traffic. Since the complete attack requires tens of thousands of queries, we performed it only against servers with a large user base such as Facebook.

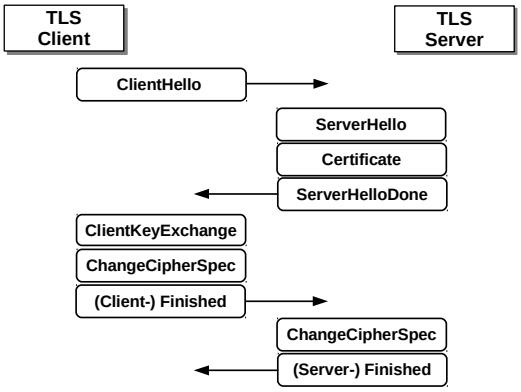

Figure 1: TLS-RSA handshake.

2 TLS-RSA key exchange

Bleichenbacher's attack is applicable to the TLS-RSA key exchange. This key exchange is used in all cipher suites having names starting with `TLS_RSA` (e.g. `TLS_RSA_WITH_AES_128_CBC_SHA`). The message flow of an RSA key exchange as implemented in TLS [34] is illustrated in Figure 1.

The TLS handshake is initiated by a TLS client with a `ClientHello` message. This message contains information about the TLS version and a list of supported cipher suites. If the server shares cipher and protocol support with the client, it responds with a `ServerHello` message indicating the selected cipher suite and other connection parameters. The server continues by sending its certificate in the `Certificate` message and signals the end of transmission with the `ServerHelloDone` message. The client then sends a `ClientKeyExchange` message containing a premaster secret that was RSA encrypted using the key included in the server's certificate. All further connection keys are derived from this premaster secret. The handshake concludes with both parties sending the `ChangeCipherSpec` and `Finished` messages. The `ChangeCipherSpec` indicates that the peer will send further messages protected with the negotiated cryptographic keys and algorithms. The `Finished` message authenticates the exchanged protocol messages.

3 Bleichenbacher's attack

Bleichenbacher's attack on SSL relies on two ingredients. The first is the malleability of RSA which allows anybody with an RSA public key to *multiply* encrypted plaintexts. The second is the tolerant nature of the RSA

PKCS #1 v1.5 padding format that allows an attacker to create valid messages with a high probability.

We assume (N,e) to be an RSA public key, where N has byte-length ℓ ($|N| = \ell$), with corresponding secret key $d = 1/e \bmod \phi(N)$. $||$ denotes byte concatenation.

3.1 RSA PKCS #1 v1.5

RSA PKCS #1 v1.5 describes how to generate a randomized padding string PS for a message k before encrypting it with RSA [25]:

1. The encryptor generates a random padding string PS, where $|PS| > 8$, $|PS| = \ell - 3 - |k|$, and $0x00 \notin \{PS_1, \ldots, PS_{|PS|}\}$.
2. It computes the message block as follows: $m = 00||02||PS||00||k$.
3. Finally, it computes the ciphertext as $c = m^e \bmod N$.

The decryption process reverts these steps in an obvious way. The decryptor uses its private key to perform RSA decryption, checks the PKCS #1 v1.5 padding, and extracts message k.

3.2 Attack intuition

Bleichenbacher's attack allows an attacker to recover the encrypted plaintext m from the ciphertext c. For the attack execution, the attacker uses an oracle that decrypts c and responds with 1 if the plaintext starts with `0x0002` or 0 otherwise:

$$\mathcal{O}(c) = \begin{cases} 1 & \text{if } m = c^d \bmod N \text{ starts with } \texttt{0x0002} \\ 0 & \text{otherwise.} \end{cases}$$

Such an oracle can be constructed from a server decrypting RSA PKCS #1 v1.5 ciphertexts.

Bleichenbacher's algorithm is based on the malleability of the RSA encryption scheme. In general, this property allows an attacker to use an integer value s and perform plaintext multiplications:

$$c' = (c \cdot s^e) \bmod N = (ms)^e \bmod N,$$

Now assume a PKCS #1 v1.5 conforming message $c = m^e \bmod N$. The attacker starts with a small value s. He iteratively increments s, computes c', and queries the oracle. Once the oracle responds with 1, he learns that

$$2B \leq ms - rN < 3B,$$

for some computed r, where $B = 2^{8(\ell-2)}$. This allows him to reduce the set of possible solutions. By iteratively choosing new s, querying the oracle, and computing new r values, the attacker reduces the possible solutions m, until only one is left or the interval is small enough to accommodate a brute force search. We refer to the original paper for more details [11].

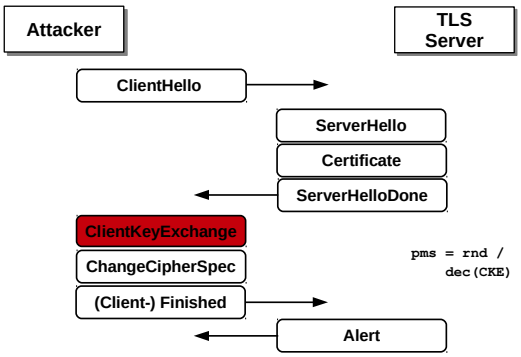

Figure 2: A vulnerable server would respond with different alert messages based on the PKCS #1 v1.5 validity. To mitigate the attack it is important that the server *always* responds with the same alert message and does not provide any information about the PKCS #1 v1.5 validity.

3.3 Countermeasures

In general the attack is always applicable if the attacker is able to distinguish valid from invalid RSA PKCS #1 v1.5 ciphertexts. To mitigate the attack, the TLS standard has defined the following countermeasure. Once the server receives a `ClientKeyExchange` message, it proceeds as follows (see Figure 2). It generates a random premaster secret and attempts to decrypt the ciphertext located in the `ClientKeyExchange` message. If the ciphertext was valid, it proceeds with the decrypted premaster secret. Otherwise, it proceeds with the random value. Since the attacker does not know the premaster secret value, he is not able to compute a valid `Finished` message. Therefore, the client `Finished` message is always responded with an alert message and the attacker cannot determine PKCS #1 v1.5 validity. See Section 9.1 for more details.

3.4 Attack performance and oracle types

In his original publication Bleichenbacher estimated that it takes about one million queries to decrypt an arbitrary ciphertext. Therefore, the attack was also named "million message attack". The attack performance varies however depending on the "strength" of the provided oracle. In general, the attack algorithm finds a new interval with every new valid oracle response. This happens if the decrypted ciphertext starts with `0x0002`. The oracle is considered "weaker" if it responds with a nega-

tive response for some decrypted ciphertexts which start with 0x0002. In this scenario, the new interval is not found and the attacker needs to issue more queries. This can happen, for example, if the implementation strictly checks the PKCS #1 v1.5 format which prescribes that the first 8 bytes following 0x0002 are non-zero, or if the implementation strictly checks the length of the unpadded key.

Bardou et al. improved the original attack and analyzed the impact of different implementations on the attack performance [7]. For example, the improved Bleichenbacher attack algorithm needs about 10,000 queries on average when using the "strongest" oracle. On the other hand, it needs about 18,000,000 queries using the "weakest" oracle.

For simplicity, in our paper we just assume two oracle types: *weak* and *strong*. The strong oracle allows one to decrypt arbitrary ciphertext in less than one million queries on average. Such an oracle can be provided by an implementation which returns true if the decrypted ciphertext starts with 0x0002 and contains a 0x00 at any position. The weak oracle results in an attack with several millions of queries and can be provided by an implementation which checks whether the 0x00 byte is located on the correct position. We use the original Bleichenbacher algorithm [11].

3.5 Creating a signature with Bleichenbacher's attack

In most of the studies, Bleichenbacher's attack is referred to as a decryption attack. A lesser noted point is that the attack allows one to perform arbitrary RSA private key operations. Given access to an oracle, the attacker is not only able to decrypt ciphertexts but also to sign arbitrary messages with server's private RSA key.

In order to create a signature with the server's private key, the attacker first uses a proper hash function and encoding to process the message. For example, when creating a PKCS #1 v1.5 signature for message M, the encoded result will have the following format [29]:

```
EM = 0x0001 || 0xFF...FF || 0x00 ||
ASN.1(hash(M))
```

hash() denotes a cryptographic hash function. The output of the hash function has to be encoded using ASN.1. The attacker then sets EM as an input into the Bleichenbacher algorithm. In a sense, he uses the to be signed message as if it were an eavesdropped ciphertext. The end result of this operation is a valid signature for M.

It is also important to mention that creating a signature is typically more time consuming than decrypting a PKCS #1 v1.5 ciphertext. The reason is that an attacker with a PKCS #1 v1.5 ciphertext can already assume that the first message is PKCS #1 v1.5 conforming. This allows him to skip the very first step from the original algorithm [11]. On the other hand, by decrypting a random ciphertext or creating a signature, the attacker cannot assume the first query is PKCS #1 v1.5 conforming. To make this first message PKCS #1 v1.5 conforming, the attacker has to apply a *blinding step* [11]. Since this step requires many oracle requests, creating a signature is much more time consuming and is only practical if a strong oracle is available.

4 Scanning methodology

The challenge of our research was to perform an effective scan using as few requests as possible, but allowing us to trigger all known vulnerabilities and potentially find new ones. For this purpose we closely modeled our first scanner after the techniques in Bleichenbacher's original publication [11] and the following research results [26, 7, 28]. This scanner performed a basic TLS-RSA handshake (see Figure 1) containing differently formatted PKCS #1 v1.5 messages located in ClientKeyExchange. With this approach, we were able to identify our first vulnerable TLS implementations. Further analysis was conducted to identify possible false positives before reporting the behavior to vendors and site operators. This manual analysis allowed us to find new issues and extend further TLS scans which we applied to the Alexa Top 1 Million list.

In the following sections we give an overview of our final scanning methodology. If possible we highlight general recommendations, which are of importance for performing related vulnerability scans.

4.1 Differently formatted PKCS #1 v1.5 messages

To trigger different server behaviors, our ClientKeyExchange messages contained differently formatted PKCS #1 v1.5 messages. For their description, consider the following notation. || denotes byte concatenation, version represents two TLS version bytes, rnd[x] denotes a non-zero random string of length x, and pad() denotes a function which generates a non-zero padding string whose inclusion fills the message to achieve the RSA key length.

Given the performance prerequisites for our scan, we carefully selected five PKCS #1 v1.5 vectors based on the previous research on Bleichenbacher attacks [11, 7, 28, 36]. Every message should trigger a different vulnerability:

1. Correctly formatted TLS message. This message contains a correctly formatted PKCS #1 v1.5

padding with 0x00 at a correct position and correct TLS version located in the premaster secret:

```
M1 = 0x0002 || pad() || 0x00 ||
version || rnd[46]
```

M1 should simulate an attacker who correctly guessed the PKCS #1 v1.5 padding as well as TLS version. Even though this case is hard to trigger (because of a low probability of constructing such a message), it is needed to evaluate the server correctness.

2. Incorrect PKCS #1 v1.5 padding. This message starts with incorrect PKCS #1 v1.5 padding bytes:

   ```
   M2 = 0x4117 || pad()
   ```

   The invalid first byte in the PKCS #1 v1.5 padding should trigger an invalid server behavior as described, for example, in the original paper [11].

3. 0x00 at wrong position. This message contains a correct PKCS #1 v1.5 format, but has 0x00 at a wrong position so that the unpadded premaster secret will have an invalid length:

   ```
   M3 = 0x0002 || pad() || 0x0011
   ```

   Many implementations assume that the unpadded value has a correct length. If the unpadded is shorter or longer, it could trigger a buffer overflow or specific internal exceptions, and lead to a different server behavior. For example, Meyer et al. showed that such a message resulted in different TLS alerts in JSSE (Java Secure Socket Extension) [28].

4. Missing 0x00. This message starts with 0x0002 but misses the 0x00 byte:

   ```
   M4 = 0x0002 || pad()
   ```

   The PKCS #1 v1.5 standard prescribes that the decrypted message always contains a 0x00 byte. If this byte is missing, the PKCS #1 v1.5 implementation cannot unpad the encrypted value, which can again result in a different server behavior.

5. Wrong TLS version. This message contains an invalid TLS version in the premaster secret:

   ```
   M5 = 0x0002 || pad() || 0x00 ||
   0x0202 || rnd[46]
   ```

   M5 should trigger an invalid behavior as described by Klíma, Pokorný and Rosa [26]. A practical example of such behavior was recently found in MatrixSSL [36]. The vulnerable MatrixSSL version responded these types of messages with an illegal parameter alert. Other messages were responded with a decryption error.

A server behaves correctly if it responds with the same alert message to any of the above messages. Otherwise, it is vulnerable to Bleichenbacher's attack. As described in Section 3.4, we say that the oracle is weak if the attacker can only identify valid messages starting with 0x0002 with a validly padded PKCS #1 v1.5 message with the 0x00 byte at the correct position (i.e., message M1 or M5). This is because of a low probability of triggering such a case during the attack. Otherwise, if the server allows the attacker to identify messages with, for example, message M3 or M4, the server provides a strong oracle and the attack can be practically exploited.

4.2 Different TLS protocol flows

We observed that several implementations responded differently based on the constructed TLS protocol flow. More specifically, we observed differences on some servers when processing a ClientKeyExchange message sent by itself versus when it was sent in conjunction with ChangeCipherSpec and Finished. We will refer to sending ClientKeyExchange alone as "shortened message flow" in the rest of the paper.

The primary example of this is F5 BIG-IP. Under certain configurations, when this device received an invalid ClientKeyExchange without further messages, it immediately aborted the handshake and closed the connection. Otherwise, when processing properly formatted ClientKeyExchange, the device waited for subsequent ChangeCipherSpec and Finished messages.

Our scans also confirmed that it is insufficient to consider only TLS alert numbers or timing as a suitable side-channel. It is also necessary to monitor connection state and timeout issues.

4.3 Cipher suites

Our initial tool implementation was trying to connect with a single AES-CBC cipher suite. During our scans we observed some servers with a limited set of cipher suites which, for example, only supported AES-GCM cipher suites. We therefore changed our tool to offer additional cipher suites by default. This increased the number of detected vulnerable servers.

In addition to new vulnerable servers, additional cipher suites allowed us to observe an interesting behavior. In some cases, the responses to various ClientKeyExchange messages varied depending on the used symmetric ciphers. For example, one of our target servers reset the TCP connection after accepting a valid PKCS #1 v1.5 formatted message when using AES-CBC cipher suites. When using AES-GCM cipher suites, the server responded with a TLS alert 51 (decrypt_error).

Invalid PKCS #1 v1.5 messages always led to a connection timeout, independently of the used cipher suite.

4.4 Monitoring different server responses

According to the TLS standard [34], servers receiving invalid `ClientKeyExchange` messages should continue the TLS handshake and always respond with an identical TLS alert. In our analyses, we observed several servers which always responded with identical TLS alerts. Some however returned an extra TLS alert when processing an invalid `ClientKeyExchange`.

In a server scan it is therefore important to not only monitor the last received TLS alert but also the content and count of received messages and socket behavior.

4.5 More variations

During our research we discovered that with slight variations like changing the cipher suite or using the shortened TLS message flow we were able to discover more vulnerable servers. A more exhaustive scan may reveal more vulnerable implementations. However, there is a very large number of potential variations to try. For example, one could try to connect with exotic cipher suites (like Camellia), extensions or new variations of message flows.

With our scan tool we attempted to find all vulnerabilities we are aware of while at the same time avoiding excessively long scans.

4.6 Performing a server scan

In summary, our server evaluation is primarily differentiated from other published techniques we are familiar with [11, 28, 36] in that we consider connection state as a side-channel signal and that we test with a non-standard message flow. Furthermore, we can detect duplicated alert messages and we enforce usage of different cipher suites to trigger invalid behavior. See Figure 3.

The oracle detection of our scanner works by first downloading a target server's certificate and using it to encrypt five `ClientKeyExchange` messages (`M1,...M5`). Each value is then sent as part of a standard handshake with a hardcoded `Finished` value. If the response was not the same for each test case, the target is presumed to be vulnerable. If the responses are identical, the server is retested using the same `ClientKeyExchange` but with an abbreviated message flow that omits `ChangeCipherSpec` and `Finished`. The responses are again compared and if any differences are spotted, the target is presumed to be vulnerable. In order to minimize false positive results due to network conditions or unreliable servers, all servers presumed to be vulnerable are retested to confirm the oracle prior to reporting the target as vulnerable. This is especially important when detecting timeout based oracles.

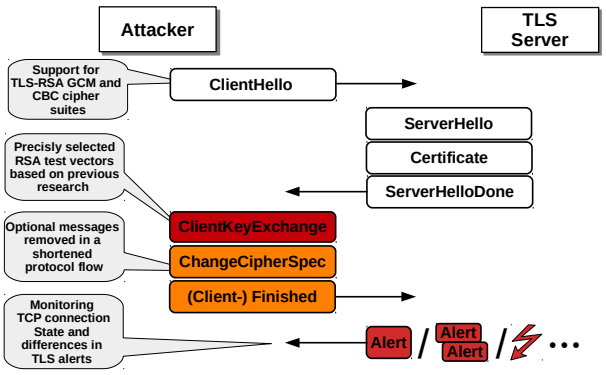

Figure 3: Our final scan considered different cipher suites, connection state, TLS alerts, and shortened protocol flow. The PKCS #1 v1.5 messages were selected precisely based on previous research [11, 7, 36].

When testing with the shortened message flow, we found it necessary to set an appropriate socket timeout for the network path between scanner and target. Tests can be performed faster with shorter timeouts but it can come at the cost of inconsistent behavior when dealing with slower hosts or network latency. In our testing, 5 seconds proved to be a reliable socket timeout for scanning over the Internet without exceeding handshake timeouts. In some environments, it may also be desirable to increase the socket timeout but setting it too high will lead to unreliable results.

5 Vulnerable implementations

The following sections present our findings and detailed behaviors of vulnerable implementations. The results are summarized in Table 1. For each vulnerable implementation the table provides information about different server responses triggered by valid and invalid `ClientKeyExchange` messages, the TLS protocol flow (full / shortened), the oracle type (strong / weak), and a CVE ID.

5.1 Facebook

During our first scans, we discovered that the main Facebook host – `www.facebook.com` – was vulnerable. The server responded with a TLS alert 20 (`bad_record_mac`) to an error in the padded premaster secret. An error in the PKCS #1 v1.5 prefix or in the padding resulted in an immediate TCP reset. We could observe a similar behavior on multiple other hosts belonging to Facebook like `instagram.com` and `fbcdn.com`.

Implementation	Server response		TLS flow	Oracle	Reference / ID
	Valid message	Invalid message			
Facebook					
1st vulnerability	20	47	full	strong	-
2nd vulnerability	20	TCP FIN	shortened	strong	-
F5					
Variant 1	TCP timeout	40	shortened	strong	CVE-2017-6168
Variant 2	One alert (40)	Two alerts (40)	full	strong	CVE-2017-6168
Variant 3	TCP timeout	40	shortened	weak	CVE-2017-6168
Variant 4	One alert (40)	Two alerts (40)	full	weak	CVE-2017-6168
Variant 5	20	80	full	strong	CVE-2017-6168
Citrix Netscaler					
with CBC cipher suites	Connection reset	TCP timeout	full	strong	CVE-2017-17382
with GCM cipher suites	51	TCP timeout	full	strong	CVE-2017-17382
Radware					
Radware Alteon	51	TCP reset	full	strong	CVE-2017-17427
Cisco					
Cisco ACE	20	47	full	strong	CVE-2017-17428
Cisco ASA	TCP timeout	TCP reset	full	weak	CVE-2017-12373
Erlang					
Erlang version 19 and 20	10	51	full	strong	CVE-2017-1000385
Erlang version 18	20	51	full	strong	CVE-2017-1000385
Palo Alto Networks					
PAN-OS	One alert (40)	Two Alerts (40)	full	weak	CVE-2017-17841
IBM					
IBM Domino	20	47	full	weak	(unfixed)
IBM WebSphere MQ	?	?	?	?	CVE-2018-1388
WolfSSL					
WolfSSL prior to 3.12.2	TCP timeout	Alert 0	shortened	weak	CVE-2017-13099
Bouncy Castle					
Bouncy Castle 1.58	`ChangeCipherSpec`	80	shortened	weak	CVE-2017-13098

Table 1: Overview of vulnerable implementations and affected servers found in our research. TLS alerts are referenced by their numbers: 10 (`unexpected_message`) 20 (`bad_record_mac`), 40 (`handshake_failure`), 47 (`illegal_parameter`), 51 (`decrypt_error`), and 80 (`internal_error`).

We created a proof of concept signature using this oracle and sent it to Facebook along with an explanation of the problem. Facebook deployed patches within a week to close the oracle. The signature can be found in Appendix A. However, after further testing with different message flows we found that the fix was not completely effective at preventing us from distinguishing between error types. If the `ChangeCipherSpec` and `Finished` were withheld, the server would wait for these messages only if the `ClientKeyExchange` decrypted properly. Certain padding errors on the other hand would trigger a TCP FIN from the server. Facebook also fixed this behavior within a week of being notified. We extended our scan tool to consider this changed strategy.

Facebook informed us that they use a patched version of OpenSSL for the affected hosts and that the bug was in one of their custom patches. We thus believe this particular variant of the vulnerability does not affect any hosts not owned by Facebook.

We have furthermore discovered other vulnerable hosts belonging to Facebook that behaved in a different way. These were running TLS stacks by F5 and Erlang. To our knowledge all vulnerable hosts owned by Facebook have been patched.

5.2 F5

Based on Facebook's encouraging responses to the first reports, we continued scanning their infrastructure and found yet another vulnerable behavior. This time, the

vulnerable behavior was observed on a server related to corporate mail which identified with a server banner indicating BIG-IP. Further scans uncovered similar behavior on other domains whose owners confirmed the devices as being from F5. Over the course of the research we discovered that F5 products could exhibit a variety of oracles depending on the specific product and configuration. Most commonly, F5 products would respond to malformed `ClientKeyExchange` with a TLS alert 40 (`handshake_failure`) but allow connections to timeout if the decryption was successful. Close analysis of F5 TLS stacks also revealed that some product configurations would send an extra TLS alert depending on the error type.

Overall, we discovered five different variations of behavior on F5 hosts. Some of these variations are weak oracles. These weak oracles still allow attacks, but they take significantly more oracle queries. With the strong variants of the F5 oracle we were again able to create proof of concept signatures.

We informed F5 and they issued a security advisory on November 17th [18]. They released patches for all supported products that were affected. CVE-2017-6168 was assigned.

5.3 Citrix

By contacting web page owners we learned that many of the implementations we identified as vulnerable were run by Citrix Netscaler devices. The Netscaler vulnerability is behaving slightly different depending on whether the connection uses a CBC or a GCM cipher suite.

For this vulnerability the signal for a malformed decryption block is a timeout. This makes practical attacks more challenging, as one needs to send a lot of messages and detect timeouts. It likely requires parallelizing the attack.

CVE-2017-17382 was assigned to this vulnerability. Citrix has published an advisory and updates for affected devices [15].

5.4 Radware

We discovered that the server used by Radware's web page – `radware.com` – was vulnerable. Messages not starting with `0x0002` were answered with a TCP reset. Other messages were answered with a TLS alert 51 (`decrypt_error`). We discovered the same issue on a host that we knew was served by a Radware Alteon device due to previous research.

We informed Radware about the issue and they released a fix with the Alteon firmware versions 30.2.9.0, 30.5.7.0 and 31.0.4.0 [32]. CVE-2017-17427 was assigned to this vulnerability.

5.5 Cisco ACE

We found that Cisco ACE load balancers were vulnerable. Different error types were answered with either TLS alert 20 (`bad_record_mac`) or 47 (`illegal_parameter`).

Cisco stopped selling and supporting ACE devices in 2013 [13]. They informed us that they will not issue a fix for this flaw. CVE-2017-17428 was assigned. Based on our scans we assume that despite being out of support for several years ACE devices are still in widespread use.

We also observed that the host `cisco.com` and several of its subdomains are vulnerable to Bleichenbacher attacks in the exact same way as the vulnerable ACE devices. Although Cisco did not reveal to us what products are used for these domains, our belief is that they are likely running out of support ACE devices within their network infrastructure.

All cipher suites supported by these devices use the RSA encryption key exchange [14], making it impossible to mitigate this vulnerability by disabling it. Users of Cisco ACE devices that need TLS support therefore cannot run these devices with a secure TLS configuration.

5.6 Erlang

We tested multiple TLS stacks in free and open source software to find further reasons for the vulnerabilities detected in our scans. We discovered that the TLS implementation in the Erlang programming language answered to different RSA decryption errors with different TLS alerts. Messages that did not start with `0x0002` were answered with a TLS alert 51 (`decrypt_error`), other errors were answered with a TLS alert 10 (`unexpected_message`).

Independently of that, we discovered several hosts used by WhatsApp (owned by Facebook) that were vulnerable in a similar way except that they answered with TLS alert 20 (`bad_record_mac`) rather than 51 in response to certain padding errors. We later learned from Facebook that these hosts were also operated using Erlang. Our assessment that these differences were due to different versions of Erlang was later confirmed by the Erlang developers. Their tests found that versions 19 and 20 answered with TLS alert 10/51 while version 18 answered with TLS alert 20/51 as observed on the WhatsApp domain.

The Erlang developers released fixes in the versions 18.3.4.7 [3], 19.3.6.4 [4] and 20.1.7 [5]. CVE-2017-1000385 was assigned for this bug.

5.7 Bouncy Castle

We shared our test tool with CERT/CC and they shared it with developers of various TLS implementations. We

learned that the Java TLS implementation of Bouncy Castle was vulnerable to a variant of ROBOT. Sending a `ClientKeyExchange` where the zero terminator of the padding was not at the right position led to a TLS alert 80 (`internal_error`). Other errors made the server send a `ChangeCipherSpec` message.

The vulnerability only appears if Bouncy Castle is using the JCE API in Java for cryptographic operations. Bouncy Castle offers an old API (org.bouncycastle.crypto.tls) and a new API (org.bouncycastle.tls). The vulnerability appears only if the new API is used in combination with the JCE API. The old API does not support the JCE API.

Bouncy Castle plans to fix this vulnerability in version 1.59. CVE-2017-13098 was assigned.

5.8 WolfSSL

WolfSSL is a TLS stack for embedded devices. With the shortened message flow, we got a timeout for a correctly formatted message and errors for all messages that had any flaw in their structure (wrong PKCS #1 v1.5 prefix, zeros in the non-zero padding, missing padding zero terminator).

This only gives a weak oracle and attacks would take very long. However, it should still be considered a security flaw. WolfSSL developers fixed this issue in version 3.13.0 [20]. CVE-2017-13099 has been assigned to this flaw.

5.9 Old vulnerabilities in MatrixSSL and JSSE

We are aware of two already known vulnerabilities in TLS stacks that have been discovered in recent years. Meyer et al. [28] have identified a vulnerability in Java / JSSE (CVE-2012-5081) that affects Oracle Java SE 7 Update 7 and earlier, 6 Update 35 and earlier, 5.0 Update 36 and earlier, and 1.4.2_38 (CVE-2012-5081). Somorovsky [36] has identified a vulnerability in MatrixSSL before 3.8.3 (CVE-2016-6883).

We found a small number of vulnerable hosts that we assume are these vulnerabilities, indicating that individuals or organizations still use unpatched versions of JSSE and MatrixSSL. In particular, one embedded device vendor was identified as using an older release of MatrixSSL in the latest firmware of some products.

5.10 Further vulnerabilities

We have identified a weak oracle in IBM Lotus Domino, distinguishable by TLS alerts 20 (`bad_record_mac`) and 47 (`illegal_parameter`). We have initially not disclosed this as IBM has not fixed this yet, after our initial disclosure it was independently discovered by others.[1] IBM released a security advisory for WebSphere MQ [21]. Due to the lack of communication from IBM we have no further information, but we believe this is a separate vulnerability.

We also learned after our disclosure that devices from Palo Alto Networks were vulnerable (CVE-2017-17841). A fix for PAN-OS is available in versions 7.1.5 and 8.0.7 [30].

Furthermore, we have identified vulnerable servers whose behavior we could not link to a specific implementation. It is often challenging to find out what products are used on hosts on the public Internet. Attempts to ask the operators usually remain unanswered and many products do not expose product or version information via the appropriate HTTP headers. The "Server" header is unreliable, as in many cases load balancers or security appliances are terminating TLS connections while the header information is generated by the HTTP server itself. The "X-Forwarded-For" header that is supposed to be used by such products is hardly used, as many developers of security appliances think that this information should be hidden.

Based on our findings we must assume that more vulnerable products exist. If we learn about them we will also add them to our web page.[2]

6 Statistics about affected hosts

We performed several scans over the Alexa Top 1 Million list for vulnerable hosts. We incrementally improved our scan strategy while at the same time informing affected web pages and vendors who started to patch their servers. Therefore there was no single point in time where we were able to identify all vulnerabilities. We want to stress that all our numbers should be considered rough estimates, as they are both over- and undercounting vulnerabilities.

We believe that two scans we performed on November 11th and November 12th give us the closest estimate for the number of vulnerable servers before our research. We did scans for all domains in the Alexa Top 1 Million both with and without a www prefix on HTTPS / port 443. It is very common that the hosts with and without www prefix are served by different TLS stacks.

We already had the shortened message flow. Apart from Facebook, none of the affected vendors had started shipping fixes at this point. Of particular importance is that this was prior to the availability of updated software for F5 appliances.

[1] `https://twitter.com/drwetter/status/943785632672907264`

[2] `https://robotattack.org/`

However these scans did not test with varied cipher suites and therefore missed some vulnerable hosts which do not present with vulnerable behavior when a CBC cipher is negotiated. These scans were also made after Facebook had already started deploying fixes among its infrastructure. Furthermore our scan tool did not yet contain a test to identify the JSSE issue (CVE-2012-5081).

While our scan tool attempts to minimize inaccuracies by validating vulnerable responses, we have observed that certain non-deterministic behavior can still be falsely identified as vulnerable.

According to these scans 22,854 hosts (2.3 %) were vulnerable among the www hosts. 17,463 hosts (1.7 %) were vulnerable among the non-www hosts. If we combine the results 27,965 hosts (2.8 %) were vulnerable on either the www or the non-www host. We assume that the reason for this low number of vulnerabilities overall is the correct mitigation implementation in OpenSSL, the most widely used TLS library.

Among the top 100 domains according to Alexa 27 (thus 27 %) were vulnerable if we combine our best scan result with previous scans of hosts that were already fixed at that point. This indicates that among high profile hosts the number of vulnerable systems is higher. The reason is a common usage of F5 products in high profile servers.

Based on the exact vulnerability we can also estimate affected vendors. We would like to stress that there's further potential for errors here, as it is possible that different vendors have the vulnerability in the same way making it difficult to accurately distinguish between vulnerable products. If we combine these two scans 21,194 hosts were vulnerable to one of the F5 variants we have seen. 5,856 hosts were vulnerable to the Citrix variant, 521 Cisco ACE, 336 Radware, 118 IBM, 6 MatrixSSL, and 5 Erlang. We also identified three additional behavior profiles which could not be attributed to any specific vendor. These behaviors were found on 923, 793, and 763 hosts, respectively.

7 Proof of concept attack

We developed a proof of concept attack that allows decrypting and signing messages with the key of a vulnerable server. The attack is implemented in Python 3. Our proof of concept is based on Tibor Jager's implementation of the Bleichenbacher algorithm.

The implementation uses the simple algorithm as described by Bleichenbacher's original work [11]. Our attack thus does not use the optimized algorithms that have been developed over the years [7]. We also did not parallelize the attack, all connections and oracle queries happen sequentially. Despite these limitations we were still able to practically perform the attack over the Internet both for decryptions and for signatures.

Our code first scans the host for Bleichenbacher vulnerabilities. We try to detect a variety of signals given by the server and automatically adapt our oracle to it.

For a successful attack we need many subsequent connections to a server. Our attack code utilizes `TCP_NODELAY` flag and TCP Fast Open where available to make these connections faster. This reduces latency and connection overhead allowing for more oracle queries per second.

We have published our proof of concept attack under a free license (CC0).

8 Impact analysis

A vulnerable host allows an attacker to perform operations with the server's private key. However, given that the attack usually takes several tens of thousands of connections it takes some time to perform. This has consequences for the impact of the attack.

TLS supports different kinds of key exchanges with RSA: Static RSA key exchanges where a secret value is encrypted by the client and forward-secrecy enabled key exchanges using Diffie Hellman or elliptic curve Diffie Hellman where RSA is only used for signing. Modern configurations tend to favor the Elliptic Curve Diffie Hellman key exchange. In a correct TLS implementation, it should not be possible for an attacker to force a specific key exchange mechanism, however other bugs may allow this.

If a static RSA key exchange is used, the attack has devastating consequences. An attacker can passively record traffic and later decrypt it with the Bleichenbacher oracle. Servers that only support static RSA key exchanges are therefore at the highest risk. We observed devices and configurations where this is the case, notably the Cisco ACE load balancers and the host `paypal.com`.

In this section we describe general applications of Bleichenbacher attacks to servers that do not support static RSA key exchange.

8.1 Attacks when server and client do not use RSA encryption

To attack a key exchange where RSA is only used for signatures, the attacker faces a problem: He could impersonate a server to a client, but in order to do this he has to be able to perform an RSA signature operation during the handshake. A TLS handshake usually takes less than a second. An attacker can delay this up to a few seconds, but not much more. Therefore, the attack needs to happen really fast. Creating a signature with a Bleichenbacher attack takes longer than decrypting a ciphertext, therefore this is particularly challenging.

However, if the client still supports RSA encryption, the attacker has another option: He can downgrade the connection to an RSA key exchange. This has previously been described by Aviram et al. [6]. We believe that in realistic scenarios it is possible to optimize the attack enough to be able to perform this, particularly for large targets that have a lot of servers. An attacker could parallelize and distribute the attack over multiple servers himself and attack multiple servers of the target. However, we have not practically tried to perform such an attack.

8.2 Attack on old QUIC

The QUIC protocol allowed a special attack scenario. Older versions of QUIC had the possibility to sign a static X25519 key with RSA. This key could then be used to run a server without the need of using the private RSA key during the handshake. This scenario has previously been discussed by Jager et al. [23] and in the context of the DROWN attack by Aviram et al. [6]. In response to the DROWN attack Google has first disabled QUIC for non-Google hosts and later changed the QUIC handshake to prevent this attack [12].

8.3 Cross-protocol and cross-server attacks

It should be noted that with Bleichenbacher attacks the attack target can be independent from the vulnerable server as long as they share the same RSA key. As shown by Aviram et al. [6] this has several practical implications. Let's assume a web service under `www.example.com` is served by a safe TLS stack that is not vulnerable. This server can still be attacked if the same RSA keys are used elsewhere by a vulnerable stack. This is possible because an attacker can use the oracle from the vulnerable server to sign messages or decrypt static RSA key exchanges with `www.example.com`. Impersonation attacks are also possible against `www.example.com` provided there is some vulnerable service using an HTTPS certificate valid for `www.example.com` and the attacker is fast enough. The most common scenario for this would be if a `*.example.com` certificate is used on the vulnerable target. We have actually observed such an example in the wild. The main WhatsApp web page – `www.whatsapp.com` – was not vulnerable. Several subdomains of `whatsapp.com` were however vulnerable and used a wildcard certificate that was also valid for `*.whatsapp.com`. These servers provided very good performance, thus we believe a parallelized attack would have allowed impersonation of `www.whatsapp.com`.

Similar attack scenarios can be imagined if different services share a certificate, a key, or have certificates that are also valid for other services. For example, a vulnerable e-mail server could allow attacks on HTTPS connections.

These scenarios show the risk of sharing keys between different services or using certificates with an unnecessarily large scope. We believe it would be good cryptographic practice to avoid these scenarios. Each service should have its own certificates and certificates that are valid for a large number of hosts - particularly wildcard certificates – should be avoided. Also private keys should not be shared between different certificates.

8.4 Attack on ACME revocation

Apart from attacks against TLS an attack may be possible if the private key of a TLS server is also used in different contexts.

An example for this is the ACME protocol [8] for certificate issuance that is used by Let's Encrypt. It allows revoking certificates if one is able to sign a special revocation message with the private key belonging to a certificate.

While this does not impact the security of TLS connections, it allows causing problems for web page operators that may see unexpected certificate validation errors.

9 Discussion

9.1 Countermeasures in TLS 1.0, 1.1 and 1.2

Bleichenbacher's original attack was published in 1998. At that time SSL version 3 was the current version of the SSL protocol. SSL version 3 was replaced with TLS version 1.0 in 1999 and this was thus the first standard that included countermeasures to Bleichenbacher's attack.

TLS 1.0 [2] proposed that when receiving an incorrectly formatted RSA block an implementation should generate a random value and proceed using this random value as the premaster secret. This will subsequently lead to a failure in the `Finished` message that should be indistinguishable from a correctly formatted RSA block for an attacker.

TLS 1.0 did not define clearly what a server should do if the `ClientHello` version in the premaster secret is wrong. This allowed Klíma, Pokorný and Rosa to develop a bad version oracle [26]. Also the countermeasures open up a timing variant of the Bleichenbacher oracle. Given that the random value is only created in case of an incorrectly formatted message an attacker may be able to measure the time it takes to call the random num-

ber generator. In TLS 1.1 [17] it was attempted to consider these attacks and adapt the countermeasures.

In TLS 1.2 [34] two potential algorithms are provided that implementers should follow to avoid Bleichenbacher attacks. These two variations contain further sub-variations, describing proposals for how to maintain compatibility with broken old implementations. However these should only be applied if a version number check is explicitly disabled. Furthermore TLS 1.2 states that the first algorithm is recommended, as it has theoretical advantages, referring again to the work of Klíma, Pokorný and Rosa [26]. It is not clear why the TLS designers decided to propose two different algorithms while also claiming that one of them is preferable. This needlessly increases the complexity even more.

The difference between the two algorithms in TLS 1.2 is the handling of wrong `ClientHello` versions. The first algorithm proposes that servers fix `ClientHello` version errors in the premaster secret and calculate the `Finished` message with it. The second algorithm proposes to always treat a wrong version number in the premaster secret as an error.

The TLS standards mention that the OAEP protocol provides better security against Bleichenbacher attacks. It was always decided however to keep the old PKCS #1 v1.5 standard for compatibility reasons.

To summarize, it can be seen that the designers of the TLS protocol decided to counter Bleichenbacher attacks by introducing increasingly complicated countermeasures. With each new TLS version the chapter about Bleichenbacher countermeasures got larger and more complex. As our research shows, these countermeasures often do not work in practice and many implementations remain vulnerable. In our opinion this shows that it is a bad strategy to counter cryptographic attacks with workarounds. The PKCS #1 v1.5 encoding should have been deprecated after the discovery of Bleichenbacher's attack.

We would like to point out that something very similar happened in TLS in terms of symmetric encryption. In 2002 Vaudenay demonstrated a potential padding oracle attack against CBC in TLS [37]. Instead of removing these problematic modes or redesigning them to be resilient against padding oracle attacks the TLS designers decided to propose countermeasures. TLS 1.2 explicitly mentions that these countermeasures still leave a timing side-channel. AlFardan and Paterson were subsequently able to show that this timing side-channel could be exploited [1].

9.2 Timing attacks

In this research we focused on Bleichenbacher vulnerabilities that can be performed without using timing attacks. We therefore point out that hosts that show up as safe in our scans are not necessarily safe from all variations of Bleichenbacher attacks. It is challenging to test and perform timing attacks over the public Internet due to random time differences based on network fluctuations.

Meyer et al. have described some timing-based Bleichenbacher vulnerabilities [28]. Given the complexity of the countermeasures in the TLS standard it is very likely that yet unknown timing variants of Bleichenbacher vulnerabilities exist in many TLS stacks.

We learned from Adam Langley that various TLS implementations may be vulnerable to timing attacks due to the use of variable-size bignum implementations. In OpenSSL the result of the RSA decryption is handled with the internal BN (bignum) functions. If the decrypted value has one or several leading zeros the operation will be slightly faster. If an attacker is able to measure that timing signal he may be able to use this as an oracle and perform an attack very similar to a Bleichenbacher attack. Other TLS libraries have similar issues.

The timing signal is very small and it is unclear whether this would be exploitable in practice. However, AlFardan and Paterson have shown in the Lucky Thirteen attack [1] that even very small timing side-channels can be exploitable.

9.3 PKCS #1 v1.5 deprecation in TLS

TLS protocol designers reacted to Bleichenbacher's research and followup research by adding increasingly complex workarounds. Our research shows that this strategy has not worked. The workarounds are not implemented correctly on a large number of hosts.

For the upcoming TLS 1.3 version the RSA encryption key exchange has been deprecated early in the design process [33]. However, as shown by Jager et al. this is not sufficient, as attacks can be performed across protocol versions [23]. If we assume that countermeasures are unlikely to be implemented correctly everywhere then the only safe option is to fully disable support for RSA encryption key exchanges.

This comes with some challenges. The alternatives to the RSA key exchange are finite field Diffie Hellman and Elliptic Curve Diffie Hellman key exchanges. There has also been a push to deprecate finite field Diffie Hellman, because clients cannot practically require safe parameters from a server. The Chrome browser developers have thus decided to disable support for finite field Diffie Hellman [10]. This leaves Elliptic Curve Diffie Hellman as the only remaining option, however, deployment of those ciphers has been delayed by patent concerns. Thus RSA encryption based key exchanges have been considered as a compatibility fallback to support old clients.

The deprecation of finite field Diffie Hellman is not

necessarily a problem here. Bleichenbacher vulnerabilities affect the server side of TLS. There is no added risks if clients still support RSA encryption based key exchanges. Therefore server operators can disable RSA encryption based key exchanges and support Elliptic Curve Diffie Hellman exchanges for modern clients and finite field Diffie Hellman for old clients.

Cloudflare informed us that on their hosts only around one percent of client connections use an RSA encryption key exchange. One of the authors of this paper operates HTTPS servers and was able to disable RSA encryption without any notable problems.

There is some indication that disabling RSA encryption on E-Mail servers is more problematic. We were able to log TLS ciphers on a mail server operated by one of this paper's authors. We identified legitimate connections to IMAP and POP3 with an RSA key exchange. By asking the affected users we learned that they all used the "Mail" app that came preinstalled on old Android 4 or in one case even Android 2 phones.

The algorithm choices on Android depend on the app. On an Android 4.3 phone we were able to observe that the Mail app connected via `TLS_RSA_WITH_AES_128_CBC_SHA`. However using the free K9Mail app a connection with an Elliptic Curve Diffie Hellman key exchange was used. Therefore in order to reduce the need to support the RSA encryption based key exchange users can switch to alternative apps that support more modern cryptographic algorithms.

Despite these challenges we believe that the risk of incorrectly implemented countermeasures to Bleichenbacher attacks is so high that RSA encryption based key exchanges should be deprecated. Considering the compatibility issues and risks we recommend that first support on the server side should be disabled. For HTTPS servers we believe that this can be done today and will only cause minor compatibility issues.

9.4 OAEP, PKCS #1 v1.5 for signatures and PSS

RSA-OAEP is an alternative to the padding provided by PKCS #1 v1.5 and provides better security for encrypted RSA. It is standardized in the newer PKCS #1 standards, the latest being version 2.2 [29]. However it was never used for TLS and it is unlikely that this is going to change.

Independent of the padding mode RSA encryption does not provide forward secrecy. Given the clear advantage of ciphers with forward secrecy enabled we believe the way forward is to use neither PKCS #1 v1.5 encryption nor RSA-OAEP in TLS. This is also the decision that has been made for TLS 1.3 [33]. RSA-OAEP may however be a better alternative for other protocols. We would like to point out that OAEP is not fully resilient to padding attacks, see Manger [27] and Meyer et al. [28] for details.

When using forward secrecy RSA can be used as a signature algorithm. This is still the most common setting in TLS, as alternatives like ECDSA have not seen widespread adoption yet. RSA signature implementations do not suffer from Bleichenbacher's attack from 1998, but the PKCS #1 v1.5 padding has another problem. In 2006, Bleichenbacher discovered a common implementation flaw in the parsing of those signatures [19]. A variation of this attack, named BERserk, was independently discovered by Delignat-Lavaud and Intel as affecting the Mozilla NSS library in 2014 [35]. While these attacks are completely independent of the RSA encryption attack from 1998, they are a good reason to deprecate PKCS #1 v1.5 both for encryption and for signatures.

RSA-PSS provides resilience against this attack and is also standardized in the latest PKCS #1 v2.2 standard [29]. TLS 1.3 will use RSA-PSS for signatures [33].

9.5 Bleichenbacher attacks in other protocols

In this research we focused on Bleichenbacher attacks against TLS. However these attacks are not limited to TLS. Jager et al. [22] have shown Bleichenbacher vulnerabilities in XML encryption, Detering et al. have shown vulnerabilities in JSON / JOSE [16] and Nestlerode has discovered vulnerabilities in the Cryptographic Message Syntax (CMS) code of OpenSSL [31].

All protocols that make use of PKCS #1 v1.5 encryption and potentially allow an attacker to see error messages are potential targets for Bleichenbacher attacks. Our recommendation to deprecate PKCS #1 v1.5 is therefore not limited to TLS – it should be avoided in other protocols as well.

9.6 Vendor responsibility

Perhaps the most surprising fact about our research is that it was very straightforward. We took a very old and widely known attack and were able to perform it with very minor modifications on current implementations. One might assume that vendors test their TLS stacks for known vulnerabilities. However, as our research shows in the case of Bleichenbacher attacks, several vendors have not done this.

There were several warnings that indicated such problems. The work from Meyer et al. in 2014 has already shown some vulnerable modern-day implementations [28]. Jager et al. have warned about the risk of Bleichenbacher attacks for TLS 1.3 [23], and were awarded with the best paper award at the "TLS 1.3 Ready Or Not"

(TRON) workshop [9]. Aviram et al. have used the idea of Bleichenbacher's attack to construct their DROWN attack [6]. It is notable that none of these publications have caused the affected vendors to test their product for such vulnerabilities.

9.7 Vulnerability detection tools

Many existing TLS vulnerability testing tools did not have tests for Bleichenbacher vulnerabilities in the past. This is likely one reason why such an old vulnerability is still so prevalent. To our knowledge TLS-Attacker[3] and tlsfuzzer[4] had tests for Bleichenbacher vulnerabilities before our research started. However, both tools are not yet optimized for usability and are likely only used by a small audience. None of the existing tools we know of had tests for the shortened message flow attacks.

We reached out to developers of several TLS testing tools prior to this publication. The developers of testssl.sh[5] developed a test that is similar to our own test tool. Kario implemented additional checks in tlsfuzzer. The test in tlsfuzzer is different to our test as it also checks for protocol violations that are not vulnerabilities. A strict interpretation of the TLS standard demands that all RSA decryption failures are answered with a TLS alert 20 (`bad_record_mac`) after the `Finished` message.

Tripwire IP360 added detection[6] for vulnerable F5 devices in ASPL-753 which was released in coordination with F5's public advisory. Generic detection of Bleichenbacher oracles will be released in coordination with this publication. SSLLabs added detection for Bleichenbacher oracles in their development version with a test similar to our own.[7]

Before our research, TLS-Attacker had implemented a basic Bleichenbacher attack evaluation with full TLS protocol flows. We extended this evaluation with shortened protocol flows with missing `ChangeCipherSpec` and `Finished` messages, and implemented an oracle detection based on TCP timeouts and duplicated TLS alerts. These new features are available in TLS-Attacker 2.2.

We encourage developers of other TLS or security test tools to include tests for Bleichenbacher attacks and for other old vulnerabilities. We hope that better test tools will detect any remaining vulnerable implementations that we have not identified during our research.

We are offering the code of our own scan tool under a CC0 (public domain) license. [8] This allows developers of other tools – both free and proprietary – to use our code with no restrictions.

[3] `https://github.com/RUB-NDS/TLS-Attacker`

[4] `https://github.com/tomato42/tlsfuzzer`

[5] `http://testssl.sh/`

[6] `https://www.tripwire.com/state-of-security/vert/return-bleichenbachers-oracle-threat-robot`

[7] `https://dev.ssllabs.com/`

[8] `https://github.com/robotattackorg/robot-detect`

10 Summary and conclusion

We were able to identify nine vendors and open source projects and a significant number of hosts that were vulnerable to minor variations of Bleichenbacher's adaptive-chosen ciphertext attack from 1998. The most notable fact about this is how little effort it took us to do so. We can therefore conclude that there is insufficient testing of modern TLS implementations for old vulnerabilities.

The countermeasures in the TLS standard to Bleichenbacher's attack are incredibly complicated and grew more complex over time. It should be clear that this was not a viable strategy to avoid these vulnerabilities.

The designers of TLS 1.3 have already decided to deprecate the RSA encryption key exchange. However, as long as compatibility with RSA encryption cipher suites is kept on older TLS versions these attacks remain a problem. To make sure Bleichenbacher attacks are finally resolved we recommend to fully deprecate RSA encryption based key exchanges in TLS. For HTTPS we believe this can be done today.

We hope that our research will help to end the use of PKCS #1 v1.5.

Acknowledgments

The authors thank Tibor Jager for providing a Python implementation of the Bleichenbacher attack, Adam Langley for feedback on QUIC and timing problems in Go TLS, Eric Mill from GSA for helping us to identify vulnerable platforms, Nick Sullivan for sharing usage numbers of RSA key exchanges from Cloudflare, Dirk Wetter and David Cooper for implementing a ROBOT check in testssl.sh and for finding bugs in our test code, Hubert Kario for finding bugs in our test code, Graham Steel, Vladislav Mladenov, Christopher Meyer, Robert Merget, Ernst-Günter Giessmann, and Tanja Lange for feedback on this paper, Ange Albertini for drawing a great logo, Garret Wasserman from CERT/CC for helping with vendor contacts, and Facebook for generous bug bounties.

Juraj Somorovsky was supported through the Horizon 2020 program under project number 700542 (FutureTrust).

References

[1] ALFARDAN, N. J., AND PATERSON, K. G. Lucky Thirteen: Breaking the TLS and DTLS Record Protocols. In *2013 IEEE Symposium on Security and Privacy* (May 2013), pp. 526–540.

[2] ALLEN, C., AND DIERKS, T. The TLS Protocol Version 1.0. RFC 2246, Jan. 1999.

[3] ANDIN, I. A. Patch Package: OTP 18.3.4.7. erlang-questions mailing list, Nov. 2017.

[4] ANDIN, I. A. Patch Package: OTP 19.3.6.4. erlang-questions mailing list, Nov. 2017.

[5] ANDIN, I. A. Patch Package: OTP 20.1.7. erlang-questions mailing list, Nov. 2017.

[6] AVIRAM, N., SCHINZEL, S., SOMOROVSKY, J., HENINGER, N., DANKEL, M., STEUBE, J., VALENTA, L., ADRIAN, D., HALDERMAN, J. A., DUKHOVNI, V., KÄSPER, E., COHNEY, S., ENGELS, S., PAAR, C., AND SHAVITT, Y. DROWN: Breaking TLS Using SSLv2. In *25th USENIX Security Symposium (USENIX Security 16)* (Austin, TX, Aug. 2016), USENIX Association, pp. 689–706.

[7] BARDOU, R., FOCARDI, R., KAWAMOTO, Y., SIMIONATO, L., STEEL, G., AND TSAY, J.-K. Efficient padding oracle attacks on cryptographic hardware. In *Advances in Cryptology – CRYPTO 2012: 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings* (Berlin, Heidelberg, Aug. 2012), R. Safavi-Naini and R. Canetti, Eds., Springer Berlin Heidelberg, pp. 608–625.

[8] BARNES, R., HOFFMAN-ANDREWS, J., AND KASTEN, J. Automatic Certificate Management Environment (ACME). Internet-Draft draft-ietf-acme-acme-08, Internet Engineering Task Force, Oct. 2017. Work in Progress.

[9] BAUMGARTEN, D. IETF-Award für Beitrag zu TLS 1.3, Feb. 2016.

[10] BENJAMIN, D. Intent to Deprecate: DHE-based cipher suites. Chromium net-dev mailing list, Mar. 2016.

[11] BLEICHENBACHER, D. Chosen Ciphertext Attacks Against Protocols Based on the RSA Encryption Standard PKCS1. In *Advances in Cryptology — CRYPTO '98* (Aug. 1998), Springer-Verlag, pp. 1–12.

[12] Add QUIC 31 in which the server's proof covers both the static server config as well as a hash of the client hello. Chromium Code Reviews, Mar. 2016.

[13] CISCO. End-of-Sale and End-of-Life Announcement for the Cisco ACE Application Control Engine ACE30 Module, Sept. 2013.

[14] CISCO. Release Note vA5(3.x), Cisco ACE Application Control Engine Module, Aug. 2014.

[15] CITRIX. TLS Padding Oracle Vulnerability in Citrix NetScaler Application Delivery Controller (ADC) and NetScaler Gateway, Dec. 2017.

[16] DETERING, D., SOMOROVSKY, J., MAINKA, C., MLADENOV, V., AND SCHWENK, J. On The (In-)Security Of JavaScript Object Signing And Encryption. In *Reversing and Offensive-oriented Trends Symposium (ROOTS)* (Vienna, Austria, Nov. 2017).

[17] DIERKS, T., AND RESCORLA, E. The Transport Layer Security (TLS) Protocol Version 1.1. RFC 4346, Apr. 2006.

[18] F5. K21905460: BIG-IP SSL vulnerability CVE-2017-6168, Nov. 2017.

[19] FINNEY, H. Bleichenbacher's RSA signature forgery based on implementation error. IETF OpenPGP mailing list, Aug. 2006.

[20] GARSKE, D. Fix for handling of static RSA padding failures. Github pull request, Nov. 2017.

[21] IBM. IBM Security Bulletin: WebSphere MQ is vulnerable to disclosing side channel information via discrepencies between valid and invalid PKCS#1 padding. ROBOT. (CVE-2018-1388), feb 2018.

[22] JAGER, T., SCHINZEL, S., AND SOMOROVSKY, J. Bleichenbacher's Attack Strikes again: Breaking PKCS#1 v1.5 in XML Encryption. In *European Symposium on Research in Computer Security (ESORICS)* (2012), pp. 752–769.

[23] JAGER, T., SCHWENK, J., AND SOMOROVSKY, J. On the Security of TLS 1.3 and QUIC Against Weaknesses in PKCS#1 V1.5 Encryption. In *22Nd ACM SIGSAC Conference on Computer and Communications Security* (New York, NY, USA, 2015), CCS '15, ACM, pp. 1185–1196.

[24] JAGER, T., SCHWENK, J., AND SOMOROVSKY, J. Practical Invalid Curve Attacks on TLS-ECDH. *20th European Symposium on Research in Computer Security (ESORICS)* (2015).

[25] KALISKI, B. PKCS #1: RSA Encryption Version 1.5. RFC 2313 (Informational), Mar. 1998. Obsoleted by RFC 2437.

[26] KLÍMA, V., POKORNÝ, O., AND ROSA, T. Attacking RSA-Based Sessions in SSL/TLS. In *Cryptographic Hardware and Embedded Systems - CHES 2003: 5th International Workshop, Cologne, Germany, September 8–10, 2003. Proceedings* (Berlin, Heidelberg, 2003), C. D. Walter, Ç. K. Koç, and C. Paar, Eds., Springer Berlin Heidelberg, pp. 426–440.

[27] MANGER, J. A Chosen Ciphertext Attack on RSA Optimal Asymmetric Encryption Padding (OAEP) as Standardized in PKCS #1 v2.0. In *Advances in Cryptology — CRYPTO 2001: 21st Annual International Cryptology Conference, Santa Barbara, California, USA, August 19–23, 2001 Proceedings* (Berlin, Heidelberg, 2001), Springer Berlin Heidelberg, pp. 230–238.

[28] MEYER, C., SOMOROVSKY, J., WEISS, E., SCHWENK, J., SCHINZEL, S., AND TEWS, E. Revisiting SSL/TLS Implementations: New Bleichenbacher Side Channels and Attacks. In *23rd USENIX Security Symposium (USENIX Security 14)* (San Diego, CA, Aug. 2014), USENIX Association, pp. 733–748.

[29] MORIARTY, K., KALISKI, B., JONSSON, J., AND RUSCH, A. PKCS #1: RSA Cryptography Specifications Version 2.2. RFC 8017, Nov. 2016.

[30] NETWORKS, P. A. ROBOT attack against PAN-OS (PAN-SA-2017-0032), Jan. 2018.

[31] OPENSSL. OpennSSL Security Advisory: CMS and S/MIME Bleichenbacher attack (CVE-2012-0884), Mar. 2012.

[32] RADWARE. CVE-2017-17427 Adaptive chosen-ciphertext attack vulnerability, Dec. 2017.

[33] RESCORLA, E. The Transport Layer Security (TLS) Protocol Version 1.3. Internet-Draft draft-ietf-tls-tls13-22, Internet Engineering Task Force, Nov. 2017. Work in Progress.

[34] RESCORLA, E., AND DIERKS, T. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246, Aug. 2008.

[35] RESEARCH, I. S. A. T. BERserk Vulnerability, Sept. 2014.

[36] SOMOROVSKY, J. Systematic Fuzzing and Testing of TLS Libraries. In *ACM SIGSAC Conference on Computer and Communications Security* (New York, NY, USA, Oct. 2016), CCS '16, ACM, pp. 1492–1504.

[37] VAUDENAY, S. Security Flaws Induced by CBC Padding - Applications to SSL, IPSEC, WTLS. In *Advances in Cryptology - EUROCRYPT 2002, International Conference on the Theory and Applications of Cryptographic Techniques, Amsterdam, The Netherlands, April 28 - May 2, 2002, Proceedings* (May 2002), vol. 2332 of *Lecture Notes in Computer Science*, Springer, pp. 534–546.

A Generated signature for Facebook

We provide a signature that signs the following text:

`We hacked Facebook with a Bleichenbacher Oracle (JS/HB).`

The text is PKCS #1 v1.5 encoded and signed with the certificate with the certificate that was used on `www.facebook.com` at the time of this research.

We provide example commands using curl, xxd and openssl that will verify this signature. We download the certificate from the crt.sh search engine in order to have a stable URL. We could alternatively get it directly from Facebook's servers via TLS, but that would stop working once the certificate expires and Facebook changes it.

This signature is using the format of OpenSSL's `rsautl` command. This command signs the raw input message and does not use the hashing that is part of PKCS #1 v1.5.

```
echo 799e43535a4da70980fada33d0fbf51ae60d32
    c1115c87ab29b716b49ab0637733f92fc985f28
    0fa569e41e2847b09e8d028c0c2a42ce5beeb64
    0c101d5cf486cdffc5be116a2d5ba36e52f4195
    498a78427982d50bb7d9d938ab905407565358b
    1637d46fbb60a9f4f093fe58dbd2512cca70ce8
    42e74da078550d84e6abc83ef2d7e72ec79d7cb
    2014e7bd8debbd1e313188b63a2a6aec55de6f5
    6ad49d32a1201f18082afe3b4edf02ad2a1bce2
    f57104f387f3b8401c5a7a8336c80525b0b83ec
    96589c367685205623d2dcdbe1466701dffc6e7
    68fb8af1afdbe0a1a62654f3fd08175069b7b19
    8c47195b630839c663321dc5ca39abfb45216db
    7ef837 | xxd -r -p > sig
curl https://crt.sh/?d=F709E83727385F514321
    D9B2A64E26B1A195751BBCAB16BE2F2F34EBB08
    4F6A9|openssl x509 -noout -pubkey > pub
    key.key
openssl rsautl -verify -pubin -inkey pubkey
    .key -in sig
```