ESD5 – Fall 2024
Lecture Notes – Lecture 9

Department of Electronic Systems
Aalborg University

October 7, 2024

Example 1 – The Largest DDoS Attack (Until Now)

In August 2023, Google mitigated the largest *Distributed Denial-of-Service* (DDoS) attack reported in the internet history, reaching a peak of 398 million requests per second. To have an idea of how large the attack was, with an overall duration of two minutes, it generated more requests than the number of article views reported by Wikipedia all over September 2023.[1] Attacks of this size have a high potential of disrupting the traffic of services and networks, causing business losses and making critical applications unavailable.

DDoS attacks work through Internet-connected machines infected with malware, which allows them to be controlled by the attacker remotely. Specifically, the group of infected machines controlled by the attacker is called a botnet. In a DDoS, the attacker aims to disrupt a server, service, or network by using the botnet to, coordinately and repeatedly, produce a huge amount of illegitimate requests. In this case, if the number of requests is high enough, the target cannot handle the traffic, resulting in a denial of service to legitimate traffic.[2] Look at Fig. 1, where we can find Eve controlling a botnet to disrupt the MyStreaming service. Accordingly, as the MyStreaming server is overwhelmed by the botnet requests, it does not have the resources available to fulfill Zoya's request, who wants to watch the new episode of her favorite series. As a result, the MyStreaming service became unreachable to Zoya.

To overwhelm the target, the attacker can explore the weaknesses of protocols in different network layers, such as the HTTP in the Application Layer (Layer 7) or the TCP in the Transport Layer (Layer 4). As an alternative, in volumetric attacks, the attacker can try to create congestion in the network that connects the target to the Internet by, for instance,

[1] `https://cloud.google.com/blog/products/identity-security/google-cloud-mitigated-largest-ddos-attack-peaking-above-398-million-rps`

[2] `https://www.cloudflare.com/learning/ddos/what-is-a-ddos-attack/`

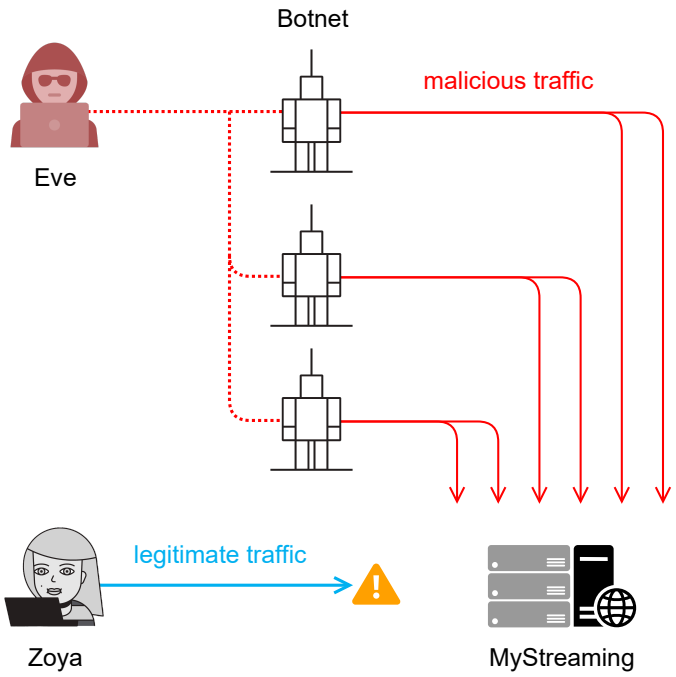

Figure 1: Example of DDoS attack, where Eve uses a botnet to flood the MyStreaming server with malicious requests and cause service disruption.

consuming all the bandwidth with malicious traffic. In the case of the DDoS attack mitigated by Google in August 2023, the attacker used the HTTP/2 Rapid Reset technique, which exploits a function of the HTTP/2 protocol that allows the client to cancel a stream/request unilaterally. This means that a client can open and immediately close a stream without any response from the target. Considering that HTTP/2 allows up to 100 concurrent streams over a single TCP connection, this function can be used to generate an indefinite number of requests in flight, as can be seen in Fig. 2.[3] Notice that, since the function to cancel a stream is also used by legit clients, to mitigate this type of DDoS attack, we cannot simply block the clients that use such a function. The mitigation depends on tracking connection statistics to identify malicious clients, as we will see next.

The mitigation of DDoS attacks essentially depends on the technique behind them. Basic solutions involve limiting the number of requests that will be accepted from a client over a specific time window (rate limiting) or adopting an application firewall that filters the requests based on a series of rules. Advanced approaches can combine multiple network statistics and fine-tuned thresholds to block malicious clients. For instance, in attacks that use the HTTP/2 Rapid Request technique, we can track the fraction of streams that are canceled by the clients, blocking the ones whose value reaches a predefined threshold. In conclusion, the most important aspect of a mitigation strategy is the between legitimate and

[3] `https://cloud.google.com/blog/products/identity-security/how-it-works-the-novel-http2-rapid-reset-ddos-attack`

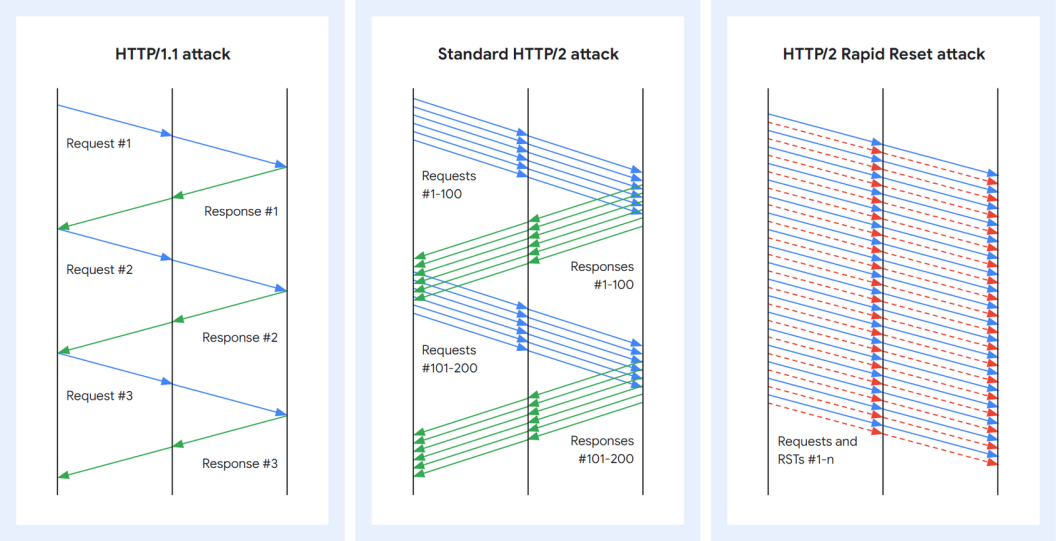

Figure 2: Request and response patterns of attacks using HTTP/1.1 and HTTP/2.

malicious traffic, preventing service unavailability for legitimate clients.

Example 2 – RSA Cube Root Attack

In implementations of the RSA cryptosystem, it is common to choose the *encryption exponent* $e = 3$ to speed up the encryption process. Despite the benefits in terms of computation, this choice makes the system susceptible to the cube root attack, which will be investigated in this example. In this attack, if *the plaintext* M satisfies $M < N^{1/3}$, where N is the modulus in the public key, an interceptor can obtain the plaintext by simply computing the cube root of the *cipher* C.

To understand how this attack works, let us consider the scenario of Fig. 3, where Zoya and Yoshi use the RSA cryptosystem to communicate ASCII characters securely. Before data transmission, in the **key establishment phase**, Yoshi shares the *public key* $(N, e) = (1002223, 3)$ with Zoya to ensure she can encrypt the messages properly. With this public key, Yoshi can obtain a plaintext from a cipher using the *decryption exponent*, or *private key*, $d = 661467$. Recall that the encryption and decryption processes in the RSA cryptosystem are respectively defined by:

$$\text{Encryption} : C = M^e \bmod N,$$
$$\text{Decryption} : M = C^d \bmod N.$$

Let us confirm if the public and private keys are valid for the RSA cryptosystem. We will do this by checking the steps to generate N, e, and d:

1) $N = 1002223$ can be written as $N = pq$, where $p = 101$ and $q = 9923$. Since 101 and 9923 are prime numbers, 1002223 is a valid modulus.

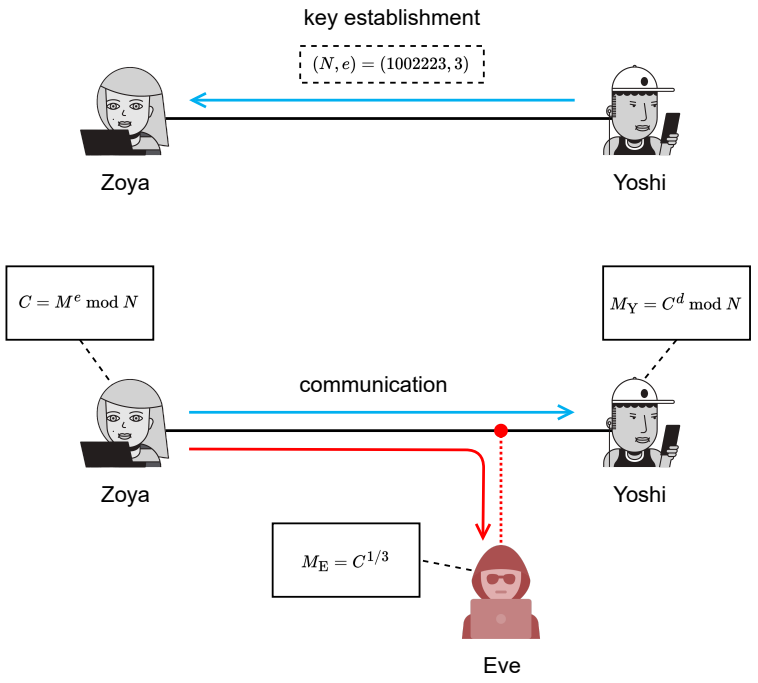

Figure 3: Example of RSA cryptosystem vulnerable to the cube root attack. The public key is $(N, e) = (1002223, 3)$, while the private key is $d = 661467$.

2) Given $p = 101$ and $q = 9923$, the common greatest divisor between $(p-1)(q-1) = 992200$ and $e = 3$ is 1, meaning that 1002223 and 3 are coprime numbers. Therefore, 3 is a valid encryption exponent.

3) With $e = 3$, $d = 661467$, $N = 1002223$, $p = 101$, and $q = 9923$ we have that $ed \bmod (p-1)(q-1) = 1$, meaning that 661467 is a modular multiplicative inverse of 3 with respect to $(p-1)(q-1)$. Hence, 661467 is a valid decryption exponent.

Since the public and private keys fulfill all the requirements, Zoya and Yoshi can now communicate securely.

In the communication phase, Zoya wants to send 3 messages comprising the characters "`A`", "`A`", and "`U`" to Yoshi. From the ASCII table,[4] we know that the character "`A`" corresponds to the byte "`0 1 0 0 0 0 0 1`" or the decimal 65, while the character "`U`" corresponds to the byte "`0 1 0 1 0 1 0 1`" or the decimal 85. This means that Zoya wants to transmit the plaintexts $M_1 = 65$, $M_2 = 65$, and then $M_3 = 85$. In Fig. 3, you might have noticed that, during communication, Eve is eavesdropping on the channel between Zoya and Yoshi to intercept the messages they are exchanging. Given the cipher C transmitted by

[4] `https://www.asciitable.com/`

Zoya, the decryption applied by Yoshi and Eve are:

$$\text{Decryption by Yoshi} : M_{\text{Y}} = C^d \bmod N,$$
$$\text{Decryption by Eve} : M_{\text{E}} = C^{1/3}.$$

In this case, the messages obtained by Yoshi and Eve are given by:

Message	**Plaintext** (M)	**Cipher** (C)	**Yoshi** (M_{Y})	**Eve** (M_{E})
A: 0 1 0 0 0 0 0 1	65	274625	65	65
A: 0 1 0 0 0 0 0 1	65	274625	65	65
U: 0 1 0 1 0 1 0 1	85	614125	85	85

Look that Eve is capable of obtaining the original plaintext by simply taking the cube root of the cipher. This happens because $M < N^{1/3}$, that is, $85 < 1002223^{1/3} \approx 100$, meaning that the RSA cryptosystem is vulnerable to the known cube root attack.

One solution to address the vulnerability of our system is to recalculate the public and private keys using another decryption exponent than $e = 3$. However, by selecting $e = 3$, we have the advantage that the encryption process can be efficiently implemented with just two multiplications. In this case, how could we address the cube root attack vulnerability without changing our keys? A solution is to use **padding**, which consists of adding data to the beginning, middle, or end of each message before encryption. Based on this strategy, Zoya can perform bit padding by adding a bit "0" at the end of each message, representing the character "A" by the bits "0 1 0 0 0 0 0 1 0" or the decimal 130, and the character "U" by the bits "0 1 0 1 0 1 0 1 0" or the decimal 170. Accordingly, the messages obtained by Yoshi and Eve become:

Message	**Plaintext** (M)	**Cipher** (C)	**Yoshi** (M_{Y})	**Eve** (M_{E})
A: 0 1 0 0 0 0 0 1 0	130	192554	130	57.74
A: 0 1 0 0 0 0 0 1 0	130	192554	130	57.74
U: 0 1 0 1 0 1 0 1 0	170	904108	170	96.69

Now, Eve cannot obtain the messages by taking the cube root of the cipher anymore, as $M > N^{1/3}$, that is, $130 > 1002223^{1/3} \approx 100$. Moreover, Yoshi can recover the original character by simply discarding the last bit, which does not carry useful information. Zoya just needs to inform him of the position of the padding bit.

The padding bit could be included at any position in the message, as long as it ensures that the plaintext satisfies $M > N^{1/3}$. Besides, considering that Yoshi will discard the padding bit, Zoya can randomize its value to prevent Eve from detecting the repetition of messages. Lets us see what happens if Zoya uses a bit "0" for the first message and a bit "1" for the second one:

Message	**Plaintext** (M)	**Cipher** (C)	**Yoshi** (M_{Y})	**Eve** (M_{E})
A: 0 1 0 0 0 0 0 1 0	130	192554	130	57.74
A: 0 1 0 0 0 0 0 1 1	131	243645	131	62.45
U: 0 1 0 1 0 1 0 1 0	170	904108	170	96.69

As you can see, despite the first two messages corresponding to the same character, the fact of using different padding bits change the cipher, making it impossible for Eve to detect the occurrence of a repetition. In this case, Zoya is using the padding bit not only to address the cube root attack vulnerability but also to obfuscate the messages.