Malice versus AN.ON: Possible Risks of Missing Replay and Integrity Protection

Benedikt Westermann[1] and Dogan Kesdogan[1,2]

[1] Q2S*, NTNU, 7491 Trondheim, Norway
[2] Chair for IT Security, FB5, University of Siegen, 57068 Siegen, Germany

Abstract. In this paper we investigate the impact of missing replay protection as well as missing integrity protection concerning a local attacker in AN.ON. AN.ON is a low latency anonymity network mostly used to anonymize web traffic. We demonstrate that both protection mechanisms are important by presenting two attacks that become feasible as soon as the mechanisms are missing. We mount both attacks on the AN.ON network which neither implements replay protection nor integrity protection yet.

1 Introduction

Anonymity networks like Tor [1] and AN.ON [2] aim to provide anonymity for their users, i.e., to hide the relation between a sender and a receiver of a message. Both networks are low-latency anonymity networks and can be used, for example, for anonymous web browsing. The low-latency requirement demands to find the right trade-off between protection and performance. Consequently, it is necessary to use only protection mechanisms that are strictly necessary concerning the attacker model. The attacker model used in Tor and AN.ON describes a local active adversary who controls a small fraction of the network.

Interestingly, Tor and AN.ON do not implement the same protection mechanisms, i.e., AN.ON neither has integrity protection nor replay protection. This raises the question about the possible risks that are introduced in AN.ON[3] by omitting the two protection mechanisms facing a local attacker.

We answer this question by presenting two different attacks. The first attack, which is referred to as *redirection attack*, exploits the lack of integrity protection and checks against a list of thousands of web sites, which web sites have been visited by a user. The result is normally a small list of web sites. The second attack, which is referred to as *replay attack*, exploits the lack of replay protection and is capable of confirming, given a small set of possible web sites that a user has visited a web site.

* *"Center for Quantifiable Quality of Service in Communication Systems, Center of Excellence" appointed by The Research Council of Norway, funded by the Research Council, NTNU and UNINETT.* http://www.q2s.ntnu.no

[3] Tor implements both protection mechanisms and is therefore not vulnerable to the presented attacks.

The paper is structured in the following way. In Section 2 we describe AN.ON's concept and the basic idea of its protocol. The attacker model and the assumptions for our attacks are introduced in Section 3. In Section 4, we present the redirection attack. The replay attack is described in Section 5. Section 6 presents related works, and in Section 7 we draw our conclusion.

2 Description of AN.ON

AN.ON uses the *cascade principle* to route the users' tra c through the AN.ON network. The cascade principle describes the selection process of the route: a user can only choose from a set of prede ned routes, so-called *cascades*. Once a user is connected to a cascade, all his messages are sent through the cascade via the same sequence of *mixes*. A cascade of length two is depicted in Figure 1. In addition to the routing over di erent nodes, messages are encrypted several times. With each mix a message passes, one layer of encryption is removed until eventually the last mix removes the nal layer of encryption and reads the address of the receiver to which the message is eventually forwarded.

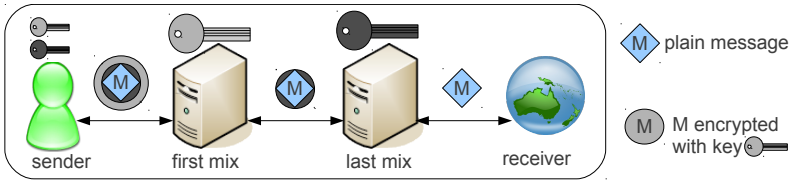


Fig. 1. A message traveling along a cascade

In order to use AN.ON, a user/sender needs to connect to one of AN.ON's cascades. The connection is established by exchanging three messages. The rst message contains a signed *descriptor* that is sent by the mix to the user. The descriptor includes, among others, the public encryption keys of the mixes in the cascade. The second message sent by the user to the mix, contains two encrypted symmetric keys. The two keys are used to encrypt the connection between the user and the rst mix. Finally the mix sends a con rmation to the user. The con rmation is a signed hash value of the keys.

When this is done, the user can request *channels*. A channel represents the (anonymous) end-to-end connection between the user and the receiver. A user can open a channel with a *channel-open* packet. It includes two session keys K^s, K^r for each mix in the cascade. The keys are encrypted with the corresponding public encryption key of the mix. In addition, the channel open packet includes the address of the end point, i.e., the receiver, of the channel which is encrypted for the last mix.

Once a channel is opened, namely the symmetric keys are exchanged and a TCP connection between the last mix and the receiver is established, a user can

exchange data with the other end point of a channel. Thereby the data is sent as payload in a *mix packet*. A mix packet is a fixed sized data structure that is used to exchange data in the cascade. The payload of a mix packet has the size of 992 bytes and it is usually encrypted in layers. AN.ON uses the *advanced encryption standard (AES)* in *output feedback (OFB)* mode as encryption scheme. By using the OFB mode, a keystream is generated that is independent of the plaintext. In order to encrypt/decrypt data, the generated keystream is simply xored with the plaintext/ciphertext.

The encrypted payload of a regular mix packet, e.g., received by a user, can be described in the following way. Let $P = p_1, \ldots, p_{992}$ be the fully decrypted payload of a mix packet and let $C = c_1, \ldots, c_{992}$ be the fully encrypted payload. Let $k_j^{r_i}$ be the j-th byte of the keystream of mix i for the packet which is generated with key K^{r_i}, then we can describe the encryption of a mix packet received by a user in a cascade of length n with the equation 1.

$$c_j = p_j \bigoplus_{i=0}^{n} k_j^{r_i} \qquad (1 \leq j \leq 992) \tag{1}$$

As pointed out in [3], AN.ON's protocol is vulnerable to replays. Currently, a replay can only be prevented if the mix stores all the keys that were used together with the mix's public encryption key. However, this is not done. The integrity of a mix packet is not protected by integrity protection mechanisms. Therefore an attacker can arbitrarily modify the encrypted packets. A more detailed description of AN.ON can be found in [3{5].

3 Attacker Model and Assumptions

For our attack we assume a local attacker who only controls the connection between the user and the first mix. Alternatively, the attacker can control the first mix in a cascade only. The attacker can add, delete, replay, and modify messages passing on the connection, but he cannot break cryptographic primitives. For both attacks, we assume that the anonymized traffic is HTTP traffic.

In addition to the above assumptions, we presume that the attacker has following capabilities:

Redirection attack: the attacker controls a web server.
Replay attack: the attacker has a small list of web sites that are likely to be a destination of a user's request, e.g., the one that is produced by the redirection attack.

4 Attack 1: Redirection Attack

The objective of this attack is to find some of the servers that the user has once visited. The attack is divided into two stages. In the first stage, the attacker tries to redirect a user to a web server that is under his control. Thereby, the attacker can greatly extend his influence. This also increases the number of

possible attacks. In the second stage, the attacker tries to extract information on previously visited servers. Here we use a *cascading style sheets (CSS)* based history recovery attack like the one presented in [6].

4.1 Redirecting the User

For our attack we utilize HTTP, which is likely to be the tunneled protocol in AN.ON. The HTTP follows the client-server paradigm. It consists of requests issued by a client and responses sent by a server. The current version is HTTP 1.1 and it is described by the RFC 2616 [7]. A typical request as well as a typical response is given in Figure 2. Figure 2(a) depicts a request and the response is shown in Figure 2(b).

```
1 GET / HTTP/1.1
2 Host: www.torproject.org
3 User-Agent: Mozilla/5.0
4 Accept: text/html
5 Accept-Language: en-us
6 Accept-Encoding: deflate
7 Accept-Charset: utf-8
8 Connection: close
```

(a) HTTP Request

```
1 HTTP/1.1 200 OK
2 Date: Tue, 06 Jul 2010
      12:46:06 GMT
3 Server: Apache
4 Accept-Ranges: bytes
5 Content-Length: 6828
6 Connection: close
7 Content-Type: text/html
```

(b) HTTP Response

Fig. 2. An example of the HTTP

The first line of an HTTP request is called the *HTTP request line* and it consists of three different parts, namely the *method*, the *request URI* and the *protocol version*. The following lines consist of key-value pairs. Thereby the key is separated by a colon from the value. The *status line* is the first line of an HTTP-response. It consists of three parts: the *protocol version*, the *status code*, and a *status message*. The following lines consist of response headers. A response header has a key and a value separated by a colon. The order of the headers is arbitrary.

For now we assume that the order of the HTTP response headers is fixed and the headers have the same order as in the example in Figure 2(b). Thus the `date` header is the first header in a response.

Normally, a web server replies with the status code `HTTP/1.1 200 OK` or `HTTP/1.0 200 OK`. We can use this knowledge to selectively modify the response.

As mentioned previously, AN.ON's designers chose AES in OFB to perform per hop symmetric encryption. Due to the missing integrity protection on any of the relevant layers in AN.ON, even an external attacker can arbitrary modify the packets passing an observed link.

The fact that the packets are encrypted various times does not complicate the attack, since each layer of encryption basically xors another keystream to

the message. The di erent keystreams can be aggregated to a single keystream[4]. Therefore we can consider for our attack AN.ON's various layers of encryption simply as one layer of encryption. The ciphertext is created by XORing the plaintext bitwise with the keystream and therefore an attacker can selectively modify single bits/bytes in an encrypted packet without modifying the remaining bytes of the eventually decrypted plaintext. For example, an attacker knows the ith byte of the plaintext, i.e., p_i, and he wants to change the ciphertext c_i such that c'_i decrypts to p'_i. He can achieve this by replacing $c_i = k_i \oplus p_i$ with $c'_i = c_i \oplus (p_i \oplus p'_i) = (k_i \oplus p_i) \oplus (p_i \oplus p'_i) = k_i \oplus p'_i$. The attacker can use this to redirect a user to a di erent server.

In this stage of the attack, the objective of an attacker is to redirect a user to his own web server. He can do so by exploiting the features of HTTP. In HTTP a status code 302 indicates a temporarily moved resource. If the browser receives such a code, it automatically redirects the user to a new resource that is given in a `location` header. Therefore, an attacker has to do two things. Firstly, he has to change the status code from 200 to 302. Secondly, he has to inject a `location` header in the HTTP response that points to his server.

For now, we assume that the rst two lines of the response are as in the example in Figure 2(b). In order to change the status code from 200 to 302, an attacker needs to change two bytes in the ciphertext. He needs to xor the ciphertext byte[5] c_{10} with $2 \oplus 3$, i.e., $c'_{10} = c_{10} \oplus 2 \oplus 3$. In addition, he needs to change the ciphertext byte c_{12} by XORing $0 \oplus 2$ to this byte. The injection of the location header, which points to the attacker's server, can be done by replacing the `date` header. The `date` header is according to our assumption the next header in the response. This header has a xed length and most parts of it are known, e.g., the year, the day, the month and most likely also the hour. Thus the attacker can exchange the `date` header by a `location` header. These two small changes are su cient to redirect a user to the server of an attacker.

Up until now we assumed that the response is as the one given in Figure 2(b). In order to evaluate if this assumption is reasonable, we collected the HTTP responses of various websites. To this end, we queried the 2000 most popular websites according to *Alexa*[6]. The HTTP response headers with its sequence number as well as the status line were stored for each website. The hostname, which is given in the Alexa database, was transformed to a URL. For example, the hostname *example.org* was transformed to *http://www.example.org*.

In 767 cases, our request resulted in a response[7] like the one shown in Figure 3 which corresponds to a percentage of 38.35 %. Here x marks the positions of the variant parts. This type of response was the most frequently seen.

This brief analysis indicates that an attacker can modify up to 31 bytes starting from the status code 200. According to the HTTP speci cation, the

[4] Due to the fact that AES in OFB is used to encrypt the connection between the mix and the user too, we can extend the argumentation in the same way.

[5] Here we ignore the header of the mix packet, but this is just a constant.

[6] http://www.alexa.com/topsites

[7] The data was collected on the 07/07/2010

```
HTTP/1.x 200 OK\r\n
Date: Wed, 07 Jul 2010 xx:xx:xx GMT\r\n
```

Fig. 3. The beginning of 38 % of the HTTP responses

status message is an arbitrary text including an empty text. Consequently, an attacker can use up to 25 bytes for the `location` header. Hereby 17 bytes are used for \Location: http://" and one byte is used for a trailing \/" to separate the domain from the path. Therefore 7 bytes remain for the domain name including the top level domain. For example, the domain abcd.xy is a suitable domain for the attack. If the header is guessed correctly, the modi cation should lead to a response presented in Figure 4. The remaining part of the `date` header can be handled by the web server as long as a \/" separates the domain from the path.

```
HTTP/1.1 \r\n
Location: http://abcd.xy/xx:xx:xx GMT\r\n
```

Fig. 4. The HTTP response header after the modification.

4.2 History Recovery

In this part we describe how an attacker can recover the browser history of a user. To do this, the attacker can exploit a well-known feature in CSS, namely the ability to display visited links di erently than non-visited links. An attack based on this feature is described in [6]. In the paper the authors show the feasibility of the attack and they estimate that at least 76% of the Internet users are vulnerable to this kind of attack.

One possibility to mount the attack is to embed a list of links to potential websites in the served HTML document. Additionally, the attacker instructs the browser to load a unique picture as background picture for each link with help of cascading style sheets. An example of such a document is given in Figure 5.

Normally, a browser parses the document and applies the style that is most speci c for each element. For the example in Figure 5, the browser would apply the style with ID `#site1`, if the site `site1.example.org` has been visited earlier. In case a user has not visited a site earlier, it applies the default style. Subsequently, the browser would issue a request to retrieve the background picture for each of the links that have been visited at least once. The URL for the referenced picture is unique due to the `GET`-parameter of the URL. Thus the attacker can log the sites that a user has visited earlier, since only the pictures of the visited sites are requested. The presented attack is feasible with the version `3.6.10` of Firefox. However, it should be noted that the browser vendors proposed and have partially implemented countermeasures for this kind of attack [8].

```
1  <style type="text/css">
2    #site1 a:visited {
3      background-image:url('/pic.php?id=1');
4    }
5    #site2 a:visited {
6      background-image:url('/pic.php?id=2');
7    }
8  </style>
9  <a id='site1' href='http://site1.example.org/'></a>
10 <a id='site2' href='http://site2.example.org/'></a>
```

Fig. 5. A example of an HTML file capable of leaking the history of a user.

4.3 From Theory to Practice

In order to mount the attack on the real AN.ON network, we intercepted and modi ed the rst packet containing an HTTP response that was received by a patched *Java AN.ON Proxy* (JAP). JAP is AN.ON's client software. The packet was selected by counting the total number of received packets. Thus it was not necessary to read the message. We modi ed the data prior to all other operations. Therefore, we simulated an external attacker. Even though we modi ed the JAP to intercept and modify the packets, it does not limit the generalizability. It is also possible to modify the packets directly on the transport layer, but it was found more convenient to patch the client to modify and intercept the packets than modifying the TCP packets on the y.

We used the patched JAP to redirect the request by modifying the encrypted packet as described in Section 4.1 and tested the attack with various sites, e.g., `http://www.youtube.com`. We opened the JAP and requested the YouTube site. As expected we were redirected to our own server. This shows that the attack is feasible and capable of redirecting a user to a di erent web server that is chosen by an attacker. With the user redirected to our own server, we mounted the previously described CSS history attack. We recovered parts of the history.

4.4 Evaluation

We demonstrate with our proof-of-concept that the attack can be mounted in the AN.ON network. In this part, we estimate the success rate of the attack.

In [6], the authors concluded that the attack succeeded at least for 76 % of the clients. There are various reasons why the attack can fail. First of all, the attack fails if a user has not visited the websites that are tested by an attacker. Another reason is that some users disable the history of the browser, or have deleted it recently.

The authors in [6] have not investigated users of anonymity networks and therefore their rate cannot be used directly for the computation of the success

rate. There are at least two reasons for this. Firstly, AN.ON's users are recommended to use a modified Firefox version, the so-called JonDoFox. The modified version disables, among others, the history of the browser such that the history attack fails. Therefore the number of users of the JonDoFox heavily influences the actual success rate of the attack. Secondly, users of anonymity networks might be more privacy aware and are therefore more likely to deactivate the history, or at least delete the history more frequently. While the effect of the latter is hard to estimate, we can take the first factor into consideration. According to the status page of the AN.ON network[8], only 14.1% of AN.ON's users are using the JonDoFox and 4.94% using "other" browsers that are not further specified. The remaining 80.96% represent well-known browsers like Firefox. For our estimation, we assume that the history attack is equally likely to work with every browser except the category "others" and "JonDoFox" with a probability of 76%. This is the success rate given in [6]. For the other two categories, we assume that the history attack always fails. Thus the probability of success is 0. Let B be the event that the attack succeeds and let $P(A_1)$ to $P(A_3)$ be the following probabilities:

$$P(A_1) = P(\text{"The user uses a well-known browser"}) = 0.8096$$
$$P(A_2) = P(\text{"The user uses JonDoFox"}) = 0.141$$
$$P(A_3) = P(\text{"The user uses another browser"}) = 0.0494$$

According to our assumption, we can express the following probabilities:

$$P(B|A_1) = 0.76$$
$$P(B|A_2) = P(B|A_3) = 0$$

By applying the law of total probability, we can compute P(B):

$$P(B) = \sum_{i=1}^{3} P(B|A_i) \cdot P(A_i) = 0.62$$

In Section 4.1, our analysis showed that in 38.35 % of the cases the redirect of user would be possible due to a correctly assumed header. Let C be the event of a successful redirect, then $P(C) = 0.38$ describes the corresponding probability.

In order to successfully mount the two stages of the attack, the redirect has to work as well as the history attack. Thus $P(B \cap C)$ is the success rate of the whole attack. Since it seems unlikely that the success of a redirect is (significantly) influenced by the success of a history attack and vice versa, we assume that both events are stochastically independent, namely $P(B \cap C) = P(B) \cdot P(C)$. Therefore the success rate of the proposed attack for a randomly chosen user in the AN.ON network for the given scenario, i.e., the attacker modifies the first HTTP response of an HTML document, is roughly 0.24.

[8] `http://infoservice.inf.tu-dresden.de:6543/status` (visited 30.09.2010)

5 Attack 2: Replay Attack

In this section, the objective of the attacker is to con rm the relation between a user and his communication partner. We focus on con rming an HTTP connection to a web server. In order to mount the attack, an attacker records messages that are sent by the user to the cascade, i.e., the rst mix in the cascade. Afterwards, he replays continuously the recorded messages into the cascade. The replayed messages are processed normally by the mixes in the cascade. Eventually they are sent to the web server that replies to the requests. The responses of the web server travel back along the cascade. The attacker measures the time that it takes to get the responses to the replayed messages. At the same time as the attacker replays the packets, he introduces an alternating arti cial load pattern at the web server, e.g., by performing various search queries, or establishing various TLS connections to the web server. Our hypothesis is that the introduced pattern should in uence the response times to replayed requests such that the attacker by observing the response times can detect the pattern. Therewith he can con rm a relationship between the user and the web server. The idea of the attack is sketched in Figure 6.

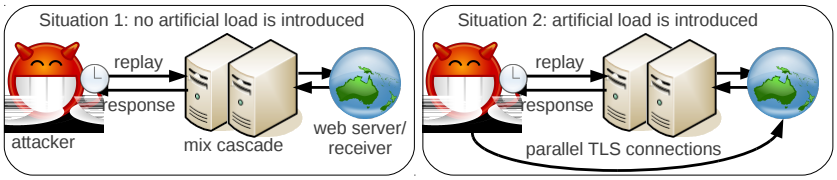


Fig. 6. An attacker has to distinguish the two situations at the delay of the responses.

In our setup, the attacker introduces the arti cial load by establishing various TLS connections in parallel to an anticipated web server. We assume that the web server has at least one port available that accepts a TLS connection, e.g., HTTPs, or SMTPs. However, there are many di erent possibilities to introduce reasonable load on a web server that do not even require the establishment of a TLS connection. The only requirement is that the request is costly for web server. Examples of such requests include, but are not limited to: search requests, dynamic page generation, or cryptographic operations.

5.1 Methodology

The proof-of-concept involves three di erent parties: a *measuring node*, a *disquieter*, and a *web server*. The task of the measuring node is to replay the previously recorded messages into the cascade and to measure the response times to the replayed requests. Additionally, the measuring node controls the disquieter. The task of the disquieter is to introduce the load pattern at a suspected web server

by establishing in parallel various TLS connections. The web server is the actual receiver of a replayed request and is attacked by the disquieter.

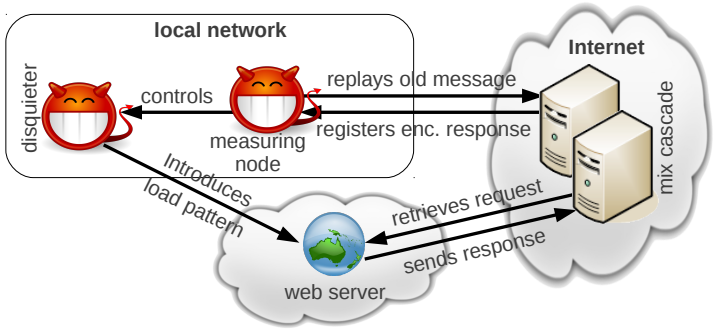


Fig. 7. Sketch of the Experimental Setup

Figure 7 depicts our experimental setup. As shown in the gure, the disquieter and the measuring node are located in the same network. Both share the same network resources. However, the network is unlikely to be a bottleneck in our experiments and it is unlikely to in uence the results signi cantly. The equipment of the used computers is given in Table 1. We con gured two di erent web servers in a typical $LAMP$ con guration. LAMP stands for Linux, Apache, MySQL and PHP. On both web servers, we installed a popular blog application, i.e., Wordpress 3.0.1. The rst server was installed in Amazon's Elastic Compute Cloud (EC2). The other web server is located in a data center in Germany. All machines were solely used for our experiments. Despite the lack of users, both web servers should represent a typical web server setup. Since our attack can be interpreted by others as denial of service attack (attacking the availability of a third party's system is against the law, e.g., in Germany), we only performed the attack on our systems.

	Measuring Node	**Disquieter**	**Web Server 1**	**Web Server 2**
CPU	Intel P8400 2.26 GHz	Intel Atom N280 1.66 GHz	Intel i7 920 2.67 GHz	Intel Xeon 5410 2.33 GHz 2 logical CPUs
RAM	4 GB	2 GB	8 GB	1.7 GB
Location	Norway NTNU's university Network		Germany in a data center	Ireland Amazon's EU EC2

Table 1. Equipment and location of the used computers

We distinguish between two types of measurements. A measurement of type 0 represents the situation in which the disquieter idles. The other situation is referred to as measurement of type 1. Here the disquieter builds up TLS connections to the anticipated web server. We alternated the two types every 30 samples. After 30 samples the measuring node instructed the disquieter to toggle the load. In total, we collected 300 samples for each type of measurement. The samples were collected in serial with a delay of 500 ms between each measurement of a sample.

We mounted the attack on several cascades with varying properties, namely *Speedpartner-ULD*, *Dresden* and *Koelsch-Rousseau-SecureInternet1*, for each of the two web servers. The *Dresden* cascade is the most popular and most frequently used cascade. It does not limit the number of users. The *Speedpartner-ULD* cascade allows at maximum 1200 users to be connected. Contrary to the first two cascades, the cascade *Koelsch-Rousseau-SecureInternet1* is a premium cascade. Here the user has to pay for the relayed traffic. Due to the involved payment protocol used on a premium cascade, a replay is not possible without controlling the first mix. Hence we modified the experiment slightly. Instead of replaying the messages into the cascade, we issued regular requests to our server with a modified version of the JAP.

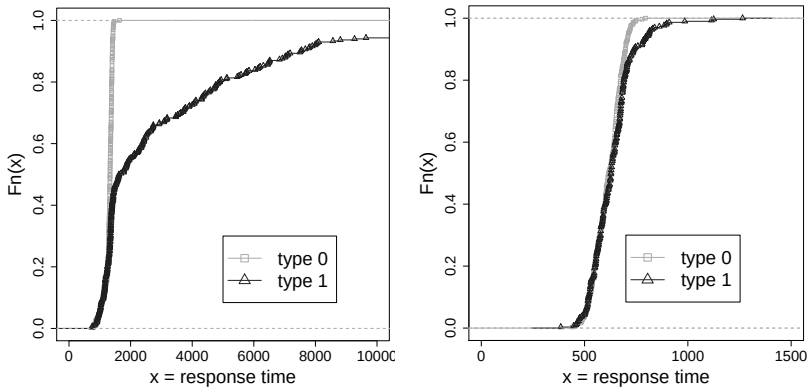
The packets for the replay were generated with the JAP. To this end, we started the JAP, connected to the cascade, issued a single HTTP request via the JAP to the web server and closed the JAP. All packets sent during the period were recorded and later on replayed into the cascade.

To replay packets, we established a TCP connection to the first mix, waited for the descriptor sent by a mix, and afterwards we replayed the first packet containing the encrypted keys. As soon as the mix had sent the third message, which is the confirmation, the time measurement starts and the remaining previously sent packets were replayed into the cascade. The time measurement stopped as soon as the same number of packets has been received as previously recorded.

In order to test if the two types of measurements are distinguishable, we used the *Kolmogorov-Smirnov* (KS) test due to its few requirements concerning the sample distribution: the only requirement is that the samples are *independently identically distributed* (iid). The KS test can be used to decide if two sets of samples stem from the same underlying probability distribution. The test is done by calculating the maximum distance between the empirical cumulative distribution functions. For our statistical analysis, we used the program R.

5.2 Measurement Results

We mounted the attack first on the not so heavily used *Speedpartner* cascade. The cascade limits the number of users on the cascade to 1200. Additionally, the bandwidth available to each user is also limited. Therefore less noise is expected. This can ease the recognition of the introduced pattern. The cascade consists of two mixes. Both are located in Germany. The first mix is located in Dusseldorf and the other one in Karlsruhe.



(a) EC2 server (b) Dedicated server

Fig. 8. The CDFs of the measurements at the Speedpartner cascade

Figure 8 shows the *cumulative distribution functions* (CDFs) of the experiment at the *Speedpartner* cascade. While the di erence of the CDFs in Figure 8(a) is clearly visible with respect to the EC2 instance, it is not as clear for the dedicated server. The CDFs for the dedicated server are shown in Figure 8(b). Here the introduced pattern has a slightly increased probability for having higher response times. However, this result is expected as the dedicated server is by far more powerful than the EC2 instance. The numeric values of the test are given in Table 2. In both cases, the KS test results in a p-value below 0.05. The value 0.05 is our rejection level for the null hypothesis, namely $F(type0) = F(type1)$. Therefore, we reject in both cases the null hypothesis. For an attacker this would mean that there is a di erence between the measurement of type 0 and type 1. Since the replays are sent to the same destination and the only variable changed by the attacker is the load on the anticipated server, it is likely that the replayed packets are sent to the tested web server.

The results for the other cascades are summarized in Table 2. Most notable is the case in which the Dresden cascade was tested together with the dedicated server as anticipated server. Although the packets were relayed to the dedicated server, it was not possible to distinguish the samples with the used methodology. The di erence between the measurement of type 0 and type 1 was probably overshadowed by the noise introduced by roughly 2500 users that were using the cascade during our measurements. In all the other cases, we were able to distinguish the two types of measurements.

Due to possible legal consequences, we did not attack productive systems. However, to test for false positive, we used some productive systems, namely *google.com*, *wikipedia.org* and *facebook.com*. We used the servers as destinations for the replayed packets while attacking one of our own servers, i.e., the EC2 instance and the dedicated server respectively. The results are given in Table 2.

		Tested server (p-values)				
cascade	**attacked server**	**EC2**	**dedicated**	**google**	**wikipedia**	**facebook**
Koelsch	ec2	$< 3 \cdot 10^{-16}$	-	0.52	0.72	0.72
	dedicated	-	$< 3 \cdot 10^{-16}$	0.72	0.40	0.08
Dresden	ec2	$< 3 \cdot 10^{-16}$	-	0.08	0.21	0.72
	dedicated	-	0.21	0.79	0.58	0.21
Speedpart.	ec2	$< 3 \cdot 10^{-16}$	-	0.45	0.85	0.79
	dedicated	-	0.04	0.72	0.25	0.90

Table 2. Summary of the p-values for all performed measurements

The absence of false positives indicates that the performance of a normally used web server is constant enough to mount the attack.

In total, we observed only a single false negative and not a single false positive. The false negative is probably caused by the load of the Dresden cascade. We assume that the load was to high to detect the introduced delay. Worth to mention is that the attack is limited. There are several cases in which the attack is likely to fail, e.g., when load balancing techniques are used. However, the attack demonstrates the risk introduced by omitting replay protection. Moreover, it is likely that the possibility of replays can help to improve other attacks, e.g. web site ngerprinting attacks [9]. This also stresses the need of replay protection even against a local attacker.

6 Related Works

There have been various timings attacks proposed during the last years. A recent example is the attack of Evans et al. [10] which is an improved attack of [11]. Here the authors identify the user selected path through the Tor network.

In [12], Hopper et al. present a timing attack that allows colluding web servers to decide whether two connections from the Tor network originates from the same user. In a second attack, the authors exploit the timing information to narrow down the location of a user of the Tor network.

Another timing attack was presented in [13]. Here the authors introduce a tra c burst on a network router and check if this in uences the data ow of a circuit. By identifying the routers in uencing the circuit, it is possible to track the full path of a circuit. While their approach works well in the controlled setting, it was harder to detect the tra c uctuations in the real Tor network.

An implementation speci c attack against Tor was proposed in [14]. Here the authors delayed cells in a way such that it is possible to in uence the number of cells in an IP packet. They used this to embed a hidden signal at the exit node of a circuit and showed that an attacker at the entry node can recover this hidden signal. Thus an attacker controlling the rst and exit node can deanonymize a user with help of this technique.

In [15], the authors describe how a local attacker can deanonymize a user with a replay attack/tagging attack in Tor. Here an attacker needs to observe

the exit as well as the entry node of the Tor network. The attack exploits that replays are only detected at the end points of a circuit. An attacker who controls the entry and the exit node can replay a packet at the first node. Eventually the replayed packet will be discovered at the exit node. Thereby an attacker can map the entry node and the exit node to the same circuit.

Another method to confirm that an entry node and an exit node participate in the same circuit was proposed in [16]. In the paper, the authors link the entry node and the exit node of a circuit with each other by correlating the $CREATE\ CELLS$ with help of their dispatch and arrival times at the corresponding nodes.

In [3], the authors inspected the cryptographic protocols used in AN.ON. Based on the flaws found in the cryptographic protocols, they proposed some attacks exploiting them.

An overview of general attacks against low-latency anonymity networks is given in [17, 18]

7 Conclusion

In this paper we exploit the lack of replay protection and integrity protection in AN.ON to mount two different attacks. Our attacks demonstrate that the lack of these protection mechanisms introduces a high risk. Thereby we only assumed a local attacker who controls the connection between the user and the first mix. Although the first attack is not capable of deanonymizing a user, e.g., the attack does not identify the web site that an attacked user intends to visit with AN.ON, it introduces a risk. It is easy-to-mount and has a good chance of success in disclosing some web sites a user has once visited. The latter can be important for follow-up attacks like the presented replay attack. This replay attack requires more resources than the redirect attack and is also more difficult to mount. However, if the attacker manage to delay the responses of the correct anticipated web server, the attack is capable of deanonymizing the user. In combination, the attacks introduce a risk that is unacceptable for a low-latency anonymity network.

Currently, the AN.ON team is working to integrate both protection mechanisms, e.g., an integrity protection mechanism is currently in the testing phase.

Acknowledgements

We like to thank Pern Hui Chia and Svein Knapskog for their valuable feedback. We also thank the JonDonym and AN.ON team for their answers to our questions and the anonymous referees for their helpful suggestions. Finally, we want to thank our shepherd Roger Dingledine.

References

1. Dingledine, R., Mathewson, N., Syverson, P.F.: Tor: The second-generation onion router. In: USENIX Security Symposium, USENIX (2004) 303–320

2. Berthold, O., Federrath, H., Köpsell, S.: Web MIXes: A system for anonymous and unobservable internet access. [19] 115–129
3. Westermann, B., Wendolsky, R., Pimenidis, L., Kesdogan, D.: Cryptographic protocol analysis of AN.ON. In Sion, R., ed.: Financial Cryptography. Volume 6052 of Lecture Notes in Computer Science., Springer (2010) 114–128
4. Köpsell, S.: Entwicklung und Betrieb eines Anonymisierungsdienstes für das WWW. PhD thesis, TU Dresden University (2010)
5. Köpsell, S.: AnonDienst - Design und Implementierung. Technical report, TU Dresden University (2004)
6. Janc, A., Olejnik, L.: Feasibility and real-world implications of web browser history detection. In: Workshop of Web 2.0 Security and Privacy 2010. (2010)
7. Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T.: Hypertext transfer protocol: Http/1.1. Internet Engineering Task Force: RFC 2616 (June 1999)
8. Stamm, S.: Mozilla security blog: Plugging the css history leak. http://blog.mozilla.com/security/2010/03/31/plugging-the-css-history-leak/ (March 2010) visited 30.09.2010.
9. Herrmann, D., Wendolsky, R., Federrath, H.: Website fingerprinting: attacking popular privacy enhancing technologies with the multinomial naïve-bayes classifier. In: Proceedings of the 2009 ACM workshop on cloud computing security, New York, NY, USA, ACM (2009) 31–42
10. Evans, N., Dingledine, R., Grothoff, C.: A practical congestion attack on tor using long paths. In: USENIX Security Symposium, USENIX (2009) 33–50
11. Murdoch, S.J., Danezis, G.: Low-cost traffic analysis of tor. In: IEEE Symposium on Security and Privacy, IEEE Computer Society (2005) 183–195
12. Hopper, N., Vasserman, E.Y., Chan-TIN, E.: How much anonymity does network latency leak? ACM Transactions on Information and System Security **13**(2) (2010) 1–28
13. Chakravarty, S., Stavrou, A., Keromytis, A.D.: Traffic analysis against low-latency anonymity networks using available bandwidth estimation. In Gritzalis, D., Preneel, B., Theoharidou, M., eds.: ESORICS. Volume 6345 of Lecture Notes in Computer Science., Springer (2010) 249–267
14. Ling, Z., Luo, J., Yu, W., Fu, X., Xuan, D., Jia, W.: A new cell counter based attack against tor. In Al-Shaer, E., Jha, S., Keromytis, A.D., eds.: ACM Conference on Computer and Communications Security, ACM (2009) 578–589
15. Pries, R., Yu, W., Fu, X., Zhao, W.: A new replay attack against anonymous communication networks. In: IEEE International Conference on Communications, IEEE (2008) 1578–1582
16. Bauer, K.S., McCoy, D., Grunwald, D., Kohno, T., Sicker, D.C.: Low-resource routing attacks against tor. In Ning, P., Yu, T., eds.: WPES, ACM (2007) 11–20
17. Back, A., Möller, U., Stiglic, A.: Traffic analysis attacks and trade-offs in anonymity providing systems. In Moskowitz, I.S., ed.: Information Hiding. Volume 2137 of Lecture Notes in Computer Science., Springer (2001) 245–257
18. Raymond, J.F.: Traffic analysis: Protocols, attacks, design issues, and open problems. [19] 10–29
19. Federrath, H., ed.: Designing Privacy Enhancing Technologies, International Workshop on Design Issues in Anonymity and Unobservability, Berkeley, CA, USA, July 25-26, 2000, Proceedings. In Federrath, H., ed.: International Workshop on Design Issues in Anonymity and Unobservability. Volume 2009 of Lecture Notes in Computer Science., Springer (2001)