Universität des Saarlandes

Naturwissenschaftlich-Technische Fakultät I

Fachrichtung Informatik

Bachelorarbeit

The MAToR Addon

Accessible Quantification of Tor Anonymity for the Tor Browser

vorgelegt von

Markus Bauer

am 13.08.2015

Begutachtet von:

Prof. Dr. Michael Backes

Dr. Christian Rossow

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides Statt, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Statement in Lieu of an Oath

I hereby confirm that I have written this thesis on my own and that I have not used any other media or materials than the ones referred to in this thesis

Einverständniserklärung

Ich bin damit einverstanden, dass meine (bestandene) Arbeit in beiden Versionen in die Bibliothek der Informatik aufgenommen und damit veröffentlicht wird.

Declaration of Consent

I agree to make both versions of my thesis (with a passing grade) accessible to the public by having them added to the library of the Computer Science Department.

Saarbrücken, _____ _____

(Datum/Date) *(Unterschrift/Signature)*

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides Statt, dass die vorliegende Arbeit mit der elektronischen Version übereinstimmt.

Statement in Lieu of an Oath

I hereby confirm the congruence of the contents of the printed data and the electronic version of the thesis.

Saarbrücken, _____ _____

(Datum/Date) *(Unterschrift/Signature)*

Abstract

Since 2004, the anonymous communication network Tor provides anonymity to its users. However, no assessment of the degree of anonymity was available to Tor users. The anonymity monitor MATor (CCS'14) was built to rigorously assess the current degree of anonymity in the Tor network. This thesis presents an addon to the Tor Browser, which uses MATor to act as a live monitor for all Tor users. The addon measures the user's anonymity based on the current network status and a user's preferences. It runs in the background of the Tor Browser and warns a user of bad anonymity guarantees or anonymity-reducing settings, thus improving a user's overall anonymity and security. The MATor addon brings current research on Tor's anonymity to Tor's users, in an easily understandable and usable way.

Contents

1. Introduction

In today's Internet, users experience a lot of privacy threats. To archive some kind of anonymity in the Internet, more than a million people use the Tor network (*"The Onion Routing network"*) [16]. The Tor network is a general-purpose anonymity solution, consisting of more than 6000 servers, provided by volunteers all around the world. The Tor Client creates anonymous and encrypted communication channels (so-called *circuits*), which consist of three proxy servers in a row. Users can route their Internet traffic through these channels, to hide their identity. A popular and easy-to-use Tor network solution is the *Tor Browser*. It allows anonymous surfing, even for laymen. But for some Tor users (like whistleblowers), anonymity is really important. Currently, these users can't assess how strong the anonymity provided by Tor is, and therefore can't assess the risk they're taking.

Related Work. In research, some work has been done in assessing the degree of anonymity of the Tor network. Based on the AnoA framework [5], the anonymity monitor MATor [6] has been proposed at CCS'14. It calculates precise bounds on Tor's anonymity under a specified thread model, based on a notion of differential privacy. MATor takes all important factors into account, like the actual network status, details of Tor's circuit path selection algorithm, and a user's settings.

However, there is a gap between theory and practice. As most research, MATor is not suitable for laymen. To use it, people would have to understand in detail how it assesses anonymity and how the formal models work. Also they would have to collect information about the Tor network, to run the calculations on them, which is quite hard because the network changes very fast. Finally, MATor outputs anonymity guarantees as numbers based on its definitions, and leave it to the user to judge them. It's hard to say from the plain anonymity guarantees if they are good or not.

Contribution. This work introduces the MATor addon, which makes MATor applicable to laymen. The MATor addon is an easy-to-use extension for the Tor Browser, which uses MATor to give anonymity guarantees for the current network state, in an easily understandable notion. The MATor addon works as live monitor for anonymity, warning if anonymity gets worse. The addon is can easily be configured for different security threads. Additionally, user actions that will harm anonymity are blocked, thus increasing user's anonymity.

2. Tor - onion routing

The Tor network is a popular way to archive anonymity [16]. It is designed to provide anonymous Internet access, by making it hard to link a user's traffic to his identity. Tor is designed to be a general-purpose, low-latency anonymity solution. It is (in theory) not restricted to a specific purpose: any TCP/IP connection can be routed through the

network. As it is low-latency, it is fast enough for real-time protocols, like web surfing for example.

The *Tor Project*, that stands behind the Tor software, provides a special browser, called the *Tor Browser* [14], to access the network in an anonymous and secure way. End-users that just want to use the network, and do not want to provide their resources to the network, normally choose the Tor Browser bundle. This way, they don't have to deal with all the pitfalls that come with the technical part of Tor (like correct and secure configuration).

According to it's website [17], Tor wants to provide anonymity in three scenarios:

- When using a specific web service, Tor does not allow that service to see the IP address, or other information about the origin of the connection. The Tor Browser extends this, as it makes sure the browser is indistinguishable from all other browser installations. Technically, this will be called *Sender anonymity*, see section 3.2.1.
- When using a surveilled Internet connection, Tor prevents the surveiller to see content or destination of the user's connections. This includes *censorship resistance* of the network (as long as the network is not censored completely). Technically, this will be called *Recipient anonymity*, see section 3.2.2.
- When routing traffic through the Tor network, Tor prevents any single network node to see who is doing what, e.g. link source and destination of a connection.

2.1. How Tor works

The Tor network consists of 6000 up to 7000 *relay nodes* (servers running the Tor software). Each of these nodes can act as a proxy for a user's communication: Instead of directly connecting to a target destination, the user establishes a connection to the proxy server, which in turn establishes a connection to the target. This way, the target can't see the origin of the connection, only the proxy server. To avoid single points of failure, Tor chains three proxy servers together. A user first connects to an *entry node* of the network, which is the first proxy. Over this connection, the user establishes a second connection inside the network, from the entry node to a *middle node*. Over this layered connection, the user establishes a third connection, from the middle node to an *exit node*, which is the one that finally establishes a connection to the target. This whole connection, routed through three intermediate nodes, is called a *circuit* (see Figure 1).

All connections (except the one between exit node and target) are encrypted. First, the user exchanges a secret key k_1 with the entry node, and uses this key to encrypt all further traffic with this node. Using this encrypted channel, the connection to the second node is established, and a second key k_2 is exchanged. The same happens for the exit node (k_3). When sending some data m, it is first encrypted with k_3 (which only the exit node knows), then with k_2, and last with k_1. The resulting *layered encryption* $c = Enc(k_1, Enc(k_2, Enc(k_3, m)))$ is sent to the first node. This node knows k_1, so it can decrypt the outer layer of the encrypted data c. It sends $c' = Dec(k_1, c) = Enc(k_2, Enc(k_3, m))$ to the middle node. The middle node knows k_2, it can remove

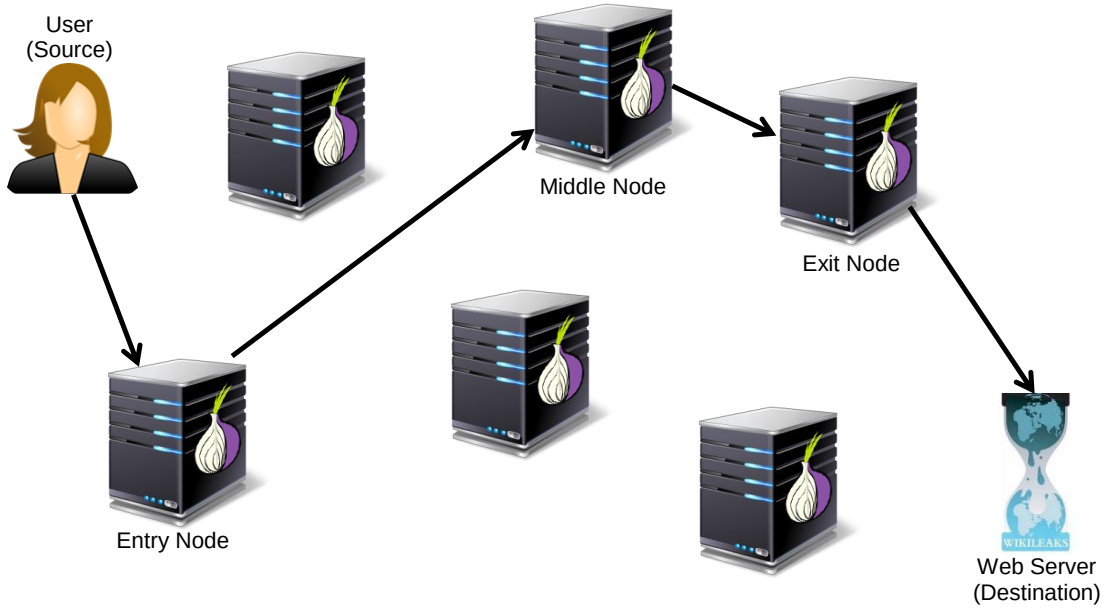


Figure 1: An example Tor circuit, using three proxy nodes

the second layer by decryption. It sends $c'' = Dec(k_2, c') = Enc(k_3, m)$ to the exit node. The exit node knows k_3, which is used to removes the last encryption layer, and recovers $m = Dec(k_3, c'')$ (see Figure 2). This cryptographic system is called *onion routing*, since the message is like the inner part of an onion, protected by several layers (of encryption). Since each node can only remove one of the layers, the message is only decrypted correctly if it passed all three nodes in the right order.

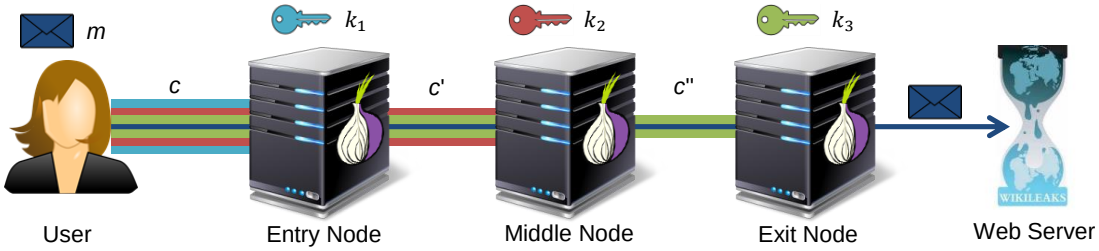


Figure 2: The layered encryption in Tor. Note that the message itself is not encrypted.

2.1.1. Tor's path selection

For the anonymity of a Tor user, it is important how the path through the network was chosen. The nodes must be chosen at random, so no adversary can guess the route a user will take through the network. However, as some nodes are faster than others, choosing the nodes uniformly at random would not archive a good performance. Choosing a

reasonably fast and secure path is not an easy task, and many different path selection algorithms have been proposed in the literature.

To choose a path, the Tor client needs a list of all nodes currently available in the network. To get this list, it asks a *directory authority*. Directory authorities are servers that contain a list of all Tor nodes known to the network. Currently, there are nine directory authorities, their addresses are hardcoded in the client software.

Upon request, a directory authority sends the list of all Tor nodes, called the *consensus file*, to the user. The consensus contains all necessary information for choosing a path, including IP addresses, bandwidth (weight), exit policy and public keys of the nodes.

Tor can now choose nodes at random. This random choice is distributed according to the *weight* of the nodes, where nodes with higher weight are chosen more likely (the probability of a node for being chosen is $\frac{\texttt{weight}}{\texttt{totalWeight}}$). The weight is given by the directory authorities, who calculate it from the available bandwidth and usage of the node. Faster servers with enough free capacity get higher weights (and will be chosen more likely) than slower nodes with less free capacity. Since some weights are more than 100000 times higher than others, around half of the nodes are almost never chosen.

As Tor works by splitting the trust on multiple parties, the path selection algorithm avoids choosing multiple nodes that might belong together. To this end, the nodes are grouped together in *families*, consisting of all nodes provided by the same person or organization. Also, the IP addresses are considered: Each circuit contains at most one node from each family, and at most one node from each `/16` ip range (all IP addresses from `a.b.0.0` to `a.b.255.255`). The ip restriction prevents using multiple nodes in the same network segment.

First, Tor chooses the exit node, because most nodes do not allow connections outside the Tor network. At the time of writing, there are around 6500 relay nodes in the network, where only around 1000 of them can act as exit node [15]. The exit nodes support different target ports, which reduces the set of possible exit nodes further. From those, an exit node is selected by random choice, distributed according to the node's weight.

Next, Tor chooses an entry node, because not every node can be an entry node. By default, Tor only chooses so-called *entry guards* as entry nodes. Entry guards are nodes that are considered specially trustworthy. At the time of writing, around 1500 of the 6500 nodes are possible guards [15]. The path selection algorithm excludes all nodes that are no appropriate entry nodes, and also all nodes that are related to the chosen exit node (in the same family or subnet). Then, a weighted random choice is done over all possible nodes, to select the entry node.

Finally, Tor chooses a middle node. There are no high requirements on this node, as each one can act as middle node. The algorithm now takes all possible middle nodes (which are much more than possible entry or exit nodes), excludes every node related to entry or exit node (by family or subnet), and again chooses one at random. After the middle node is chosen, the Tor client can start establishing it's connection.

2.1.2. Tor's security

The Tor network strives to provide *unlinkable anonymity* on the network level to its users. For any connection, it should not be possible to link the source to its target. However, this applies only to the connection itself, since Tor does not anonymize the content of the connections. If a user sends identifiable data over the anonymous connection, Tor can't help. In this context, it's important to note that the "last mile" between exit node and destination is not encrypted by Tor, which might lead to privacy issues.

Tor archives trust by splitting trust over multiple parties. In theory, only one honest node in a circuit is enough to anonymize the whole circuit. Since each node is used by multiple users, it also has multiple incoming and outgoing connections. Even if bad nodes in a circuit see both the incoming and the outgoing connection of a honest node, they are not able to link these two together, since there are many possibilities.

However, this does not hold in practice. For an attacker that wants to deanonymize a user, it suffices to capture traffic at the source and the target of a circuit, as he can link the traffic based on traffic patterns, by performing a so-called *traffic correlation* attack.

On first glance, the traffic at the source and the target of a circuit are not related: Since they are both encrypted using different keys (see section 2.1), the traffic content looks completely unrelated. However, the amount and timing pattern of the traffic is related. The pattern of data packet size, pauses between two packets and data direction changes is roughly equal on all positions of the circuit, since Tor is a low-latency network. By comparing this pattern at the source and the target of a circuit, an attacker can see if two connections belong to the same circuit or not. Research showed that this kind of attack works quite well in the Tor network [10, 12, 13].

When considering traffic correlation attacks, a single honest node is not enough to provide anonymity. If the middle node is honest while entry and exit node are not, the user's anonymity can be broken. Therefore, at least one of the entry or the exit node must be honest in order to get an anonymous connection.

In some scenarios, even this is not enough. Tor also strives to provide anonymity in the case of a web service that wants to deanonymize its users. If the owner of the web service also controls the entry node, he can apply traffic correlation attacks to the entry node's traffic and the incoming traffic of his website, and link these two together, thus deanonymizing the user. On the other hand, Tor wants to provide anonymity when using a surveilled Internet connection. Here, the surveiller can capture all traffic at the source of the circuit. If he has access to the exit node of the circuit, traffic correlation can be used to deanonymize the user. In these two cases, even a single dishonest node at the wrong position can harm a user's anonymity.

The worst case is a user with surveilled Internet connection, that visits a web service trying to track him. If the web service works together with the surveiller, he can apply traffic correlation, since he has traffic from the source (Internet connection) and the target (web service) of the circuit, thus breaking Tor without any dishonest nodes in a circuit. In this worst case, Tor can't provide secure anonymity any more.

2.2. The Tor Browser

When people want to use the Tor network, they normally use the Tor Browser [14]. It provides a simple, end-user friendly way to stay anonymous while browsing the web. The Tor Browser is a portable, standalone application, that runs on most current (desktop) operating systems (Windows, Linux and OSX). It can be launched without installation, and does not leave traces of the visited sites on the computer.

Under the hood, the Tor Browser is a modified version of the *Mozilla Firefox* browser. The current version 5.0 is based on Firefox 38 ESR (the latest long-term support version of Firefox). When using the browser, the installed *TorLauncher* extension runs the Tor client software in the background, while the *TorButton* addon changes the proxy settings in a way that all connections will be made through the Tor client.

To increase anonymity further, the Tor Browser implements some other measures that aim at providing *unlinkable anonymity* (different browsing sessions of the same user can't be linked together).

To reach this goal, the browser tries to make all instances indistinguishable from each other. First, the private-browsing mode of Firefox is enabled by default, so that no cookies, histories or cache files are preserved when the last window is closed. After each start, the browser profile is fresh and completely equal to the profiles of all other Tor Browser installations. This also helps in case somebody gets access to the Tor Browser installation, as no information about the visited sites persists. The only exceptions are bookmarks and downloaded files. Additionally, the Tor Browser does not write anything outside of his installation directory. If a user deletes this folder, no traces of the Tor Browser remain.

In addition, the Tor Browser tries to prevent *browser fingerprinting* as good as possible. Normally, every browser installation has some properties that distinguish it from others, like the version number, operating system, timezone, screen resolution, the language of the user, installed addons and fonts, or even the used hardware [20]. The Tor Browser unifies most of this properties, for example, it uses fixed values for the screen resolution, the operating system (Windows 7), the version, the timezone (UTC) and more. Also, the Tor Browser by default blocks reading from HTML5 canvas elements, since it can be used to fingerprint hardware.

Tor Browser comes with the pre-installed *NoScript addon*, that aims at preventing anonymity breaks by Javascript code on webpages. Using this addon, the Tor Browser can block some Javascript optimizations and functions, scripts from unverified (non-ssl) pages, or even all scripts.

Much worse for anonymity (and security) than Javascript are plugins like Java and Flash. By default, the whole plugin system is disabled, and it is strongly discouraged to enable it. Simple browser extensions are still allowed, but not recommended.

On the network level, the browser tries to enforce the use of HTTPS, whenever it's available, by using the *SSL Everywhere* addon of the EFF. This way, bad exit nodes should be prevented from learning or manipulating the communication. Also SSL stripping should be prevented. If the *SSL Observatory* setting is enabled, even *man-in-the-middle* attacks using faked certificates might be detected.

There are many other differences to Firefox, that are designed to increase both security and anonymity of the user. For example the Tor Browser warns the user if he downloads a file, since opening a downloaded file in an external program can result in network requests outside the Tor network, that break the anonymity. Also, the default search engine is changed from Yahoo to the more privacy-friendly engines *Startpage*, *DuckDuckGo* and *Disconnect.me*.

Overall, the Tor Browser tries to improve anonymity on all levels, but still it is easy to use, even for non-technical users.

3. MATor - guaranteed anonymity

MATor is a tool for measuring anonymity of the Tor network, developed at Saarland University, and first shown at the CCS '14 [6]. It takes Tor network data as input (the Tor consensus), and outputs some values between 0 and 1, that indicate how good anonymity is for the given network. MATor works only on the network level, not considering the content of messages send through that network.

MATor is based on an extension of the AnoA framework (IEEE CSF 2013). The AnoA framework [5] is designed for analyzing anonymity of anonymous communication networks. The used anonymity notion is similar to *differential privacy*. Given for some factor ε, MATor calculates the probability δ that the used anonymity definition is broken. ε is a fixed value.

3.1. MATor's anonymity definition

MATor works in a setting that is defined similar to typical cryptographic definitions. It's a "game" between the (simulated) Tor protocol, and an attacker that wants to deanonymize users.

Basically, the *attacker* has complete knowledge of the Tor network up to an uncertainty of one bit, which is formalized in two scenarios. For example, a user requests either page A or page B through the given network, which results in two possible connections. The attacker wins the game if he distinguishes the scenarios.

The game works like this: The attacker can freely choose the network entities that communicate, for both sources and destinations. He also knows the (open-source) path selection algorithm, but not the concrete paths the users select. We also assume that traffic correlation attacks (see 2.1.2) always work on Tor, if the attacker can fulfill their requirements. These assumptions allow us to calculate worst-case guarantees.

First, the attacker chooses some Tor nodes he wants to compromise, up to a given number of nodes, that depends on the strength of the attacker (see 3.3). He will see all data passing that nodes, including their direct source and direct destination. The attacker does not have further abilities, like active attacks on other nodes. Next, the user(s) choose their circuits and one of the possible scenarios happens. If the information the attacker can gain from his nodes suffices to guess which scenario happened, the attacker wins.

MATor calculates the maximal chance that a perfect-playing attacker wins. This is the anonymity value (δ) for a given setting.

3.2. MATor's anonymity notions

From this basic definition, AnoA defines three anonymity notions, that describe different settings in which Tor should provide anonymity: Users visiting a dishonest web service, users using a surveilled Internet connection, and attackers that simply capture traffic from the network.

3.2.1. Sender anonymity

Sender anonymity is the anonymity of users against a bad web service, that tries to deanonymize it's visitors. For example, this is interesting for users searching for cancer treatment, or visiting a police websites [19], which was considered a "suspicious action" in Germany. From 2001 on, the federal police office ("Bundeskriminalamt") has surveilled it's own manhunt homepage, and surveilled people visiting it too often.

In the sender anonymity setting, there are two users U_1 and U_2 using the network. One of these users visits a website A, which is controlled by the attacker. That means, the attacker always sees the exit node of the user's circuit.

Since we assumed that traffic correlation always works, the attacker will most likely try to corrupt the most likely entry nodes. If the user uses a corrupted entry node, traffic correlation can be used to connect the incoming connection at the website and the relayed traffic at the entry node, which allows the attacker to see which one of the two users visits the site, the attacker wins directly.

3.2.2. Recipient anonymity

Recipient anonymity is the anonymity of users with a surveilled Internet connection. For example, this is the case when using Tor in a country like China, at work, or in an unprotected wireless network. For these observers, it clearly should be indistinguishable if a user visits for example Wikipedia or Wikileaks.

In this setting, there is a single user U, that visits either site A or site B. The attacker always sees the outgoing connections of the user, e.g., between user and entry node. Of course he can't decrypt the encrypted connection, but he can use his observations for mounting a traffic correlation attack.

Most likely, an attacker in this case will try to corrupt the most likely exit nodes. When he sees a connection from one of his exit nodes to A or B, he knows the target, and using traffic correlation, he can link it to the user.

3.2.3. Relationship anonymity

Relationship anonymity means the protection against malicious Tor nodes in general. In this case, the surveillance is only focused on the network, trying to deanonymize

connections within the Tor network. For example, an intelligence agency might try to see who talks to whom through the network.

This setting defines two users U_1 and U_2, and two sites A and B, that are chosen by the attacker. Now there are two possible scenarios. In the first scenario, U_1 can connect to A or U_2 can connect to B. In the second scenario, U_1 can connect to B or U_2 can connect to A. In both scenarios, only one of the possible connections is actually made.

In this setting, the attacker has no additional power despite the corrupted network nodes. Most likely, he will try to corrupt entry and exit nodes, hoping that the circuit is built over his entry and exit node, which allows him to apply traffic correlation and see who talks to whom. This attack is much harder, so the anonymity guarantees for relationship anonymity are better that for the other two settings.

3.3. MATor attackers

All MATor attackers have the ability to choose some Tor nodes that they want to corrupt. The attackers have different strength and powers, that determine which and how many nodes can be corrupted. In addition, each attacker has some setting-given abilities (like reading all traffic of a user's Internet connection in the recipient anonymity setting).

- **k-of-N-attackers** can corrupt at most k nodes, no matter how big, fast and trusted they are. These attackers are the simplest one for calculations, but not very realistic.
- **Bandwidth attackers** can corrupt as many nodes as they want, as long as these nodes do not have a total bandwidth higher than some boundary value bw. These attackers can be more realistic for some threat scenarios, as more bandwidth costs more money.
- **Money attackers** have a budget of some money, that can be used to buy servers within the network. For example, MATor can use the server pricing of amazon's web services to calculate these costs. Using such an attacker can answer questions like "How expensive is it to break a user's anonymity?".
- **Geographical attackers** control all nodes within some region (like a country). This might be a realistic model for some states that are trying to break Tor. This attacker type might be combined with other models (for example: all nodes within a country and some additional nodes with bandwidth $\leq bw$).

Internally, all attackers are represented by a *costmap* and a budget n. The costmap maps every node r to some $costs(r) \in \mathbb{R}_0^+$. The attacker can corrupt as many nodes of his choice as he wants, as long as the sum of their costs is does not exceed n. In MATor, you can use almost arbitrary costmaps (by defining a cost function). All attacker types can be formalized using costmap and budget:

- For **k-of-N-attackers**, $costs(r) = 1$ for all nodes, and budget $n = k$.

- For **Bandwidth attackers**, $costs(r)$ is equal to the bandwidth of the node r, the budget is the maximal allowed bandwidth bw.
- For **Money attackers**, $costs(r)$ is equal to the estimated server costs of a relay node (based on location, speed and traffic). The budget n is exactly the money budget.
- For **Geographical attackers**, $costs(r) = 0$ for all nodes r within the attacker's geographic region, $costs(r) = \infty$ otherwise. Budget n is 0.

3.4. Anonymity guarantees

Given an attacker A (in form of *costs* and budget n), and a network configuration, MATor calculates the maximal probability of the attacker to win in the game from 3.1.

In order to do so, MATor calculates for each node the probabilities to be entry, middle, or exit node of the connections, according to the path selection algorithm (2.1.1) and the user's settings. These probabilities might be different for the two connection possibilities. For example in the recipient anonymity setting, if the two visited servers require different ports, the possible exit nodes will be very different.

Next, the anonymity impact $\delta(r)$ is calculated for each node r. It gives the probability that corrupting this node leads to a success for the attacker. For example in the recipient anonymity setting, $\delta(r)$ will mostly be the same value that the exit node probability of r, since corrupting the exit node results in a guaranteed success, and corrupting entry or middle node only helps in some cases.

The best-playing attacker now chooses as many nodes as he can, to maximize his success probability $\delta \approx \sum_i \delta(r_i)$, while not exceeding its cost budget: $\sum_i costs(r_i) \leq n$. This is exactly the well-known *knapsack problem*, which is *NP*-hard. MATor uses several algorithms to solve this. In the easiest case, the k-of-N attacker, MATor simply chooses the k nodes with the biggest $\delta(r_i)$ values, since the costs are equal for all. In the other cases, it uses the GNU Linear Programming Kit to obtain an optimal solution to this problem. Since computing an optimal solution takes some time, MATor comes with a "fast mode", where a dynamic programming based heuristic is used to solve the problem.

Finally, we get a value δ which is guaranteed to be the best chance an attacker can have in the anonymity game. The lower this value is, the better is the anonymity of the analyzed network and setting.

4. The MATor Addon

MATor measures anonymity based on historical data. But it's more interesting to apply this anonymity measurement on the current Tor network data, building a live monitor for anonymity. The MATor addon is such a live monitor, it measures Tor's anonymity from within the Tor Browser, working in the background, without any need for additional data.

The addon is mainly dedicated to laymen without technical background. Everybody using the Tor network should be able to use and basically understand the tool. For users that are curious or have some understanding on the topic, the addon provides advanced functionalities.

The typical addon user is clearly different from the typical MATor user. By design, the original MATor was developed for scientific research, which doesn't require usability (see Figure 3). The original tool is a console application that requires a lot of data and some computational power, to give out results as exact as possible. On the other hand, the live monitor has to be easy to use and fast even on older machines. In the addon, we can only use the data that Tor fetched to operate, and calculations shouldn't take longer than a few seconds. If the addon imposes too much workload on the cpu, the users would become annoyed.

Figure 3: The original MATor

The MATor version that ships inside the addon it adjusted to this setting. It does not use resource-heavy, exact algorithms, but uses highly-optimized heuristics instead. These methods deliver reasonably precise results in very little time, that are still accurate enough for this use case. The impact on the precision of all optimizations together is normally below 0.1% (absolute).

4.1. Installation

Installing the addon is really simple. First, the user needs a supported operating system and Tor Browser. Supported are Windows, Linux and MacOS, and the Tor Browser versions starting from the current version (version `5.0`). Next, the user simply clicks on a download link (or drags the downloaded `.xpi` file into the browser) and confirms the security warning. The MATor addon now installs and starts working without any further action, as the user can see on the new icon in the toolbar (see Figure 4).

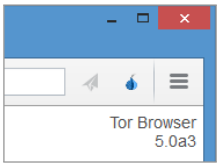


Figure 4

4.2. General functionality

While surfing the network, the addon watches out for new Tor consensus data. As soon as the Tor client loads new network information, the MATor addon calculates the anonymity guarantees for the current network state. The user can see the current guarantees by clicking on the icon (Figure 5). They are presented in a way as intuitive as possible: As colored symbols ranging from green over yellow to red. This notion is

easily understandable for everyone, even without any further background.

The exact values are presented in a tooltip. Here, the addon uses real "anonymity guarantee" values, not the more abstract δ-values from MATOR. The addon shows the user's chances of staying anonymous (called *anonymity guarantees*), not an attacker's chance to deanonymize the user. The anonymity guarantee is $1 - \delta$, given as percents between 0% and 100%. Users without any background knowledge understand this notion easier, since they do not have to know about "attackers", or "attacking scenarios" in the first place. Also, the intuition "higher anonymity (values) are better" works. For the users, it's clear that 100% anonymity is almost perfect.

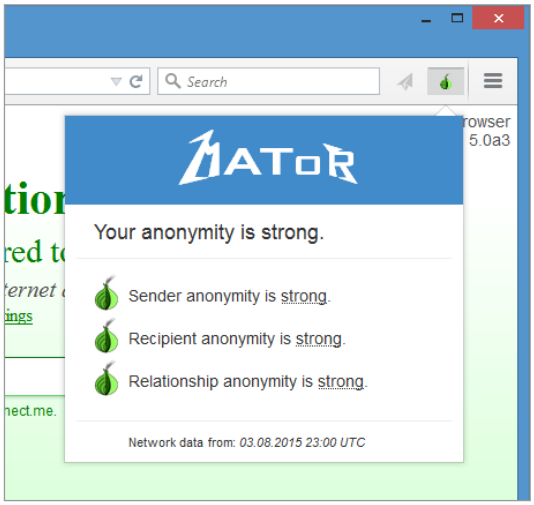


Figure 5: Main addon panel

Typically the addon stays silently in the background. After the start and every hour, when new network data arrives, it requires some cpu power, without freezing the Tor Browser. As this occurs only once an hour, and only takes a few seconds (7 - 15 seconds, depending on hardware and settings), users should not be annoyed.

4.3. Warnings

When running in the browser, the addon always monitors the current anonymity of the network. This can also be used to provide some early warning system against large-scale attacks on the network, which might increase an attacker's chance to deanonymize a user. Depending on the exact type of the attack (and the selected attacker), the anonymity guarantees will get worse if the network changes in an awkward way. For example, if multiple big Tor nodes drop out at the same time, this might be caused by a systematic DDoS attack. In this case the overall anonymity will decrease, since the remaining nodes are chosen more likely, and a fixed number of corrupted nodes has a bigger chance of being chosen. The MATOR addon tries to detect this by watching the current anonymity guarantees, and shows a warning popup, that there's something going on in the network (Figure 6).

Figure 6: Anonymity dropped

This way, the addon helps at choosing the right time when using Tor. There are times when it's much easier to break anonymity than other times, and the addon warns the user when it's such a bad time. Warnings can occur on three levels, ranging from "suspicious" to "danger", depending on how bad the current guarantees are. These warnings are easily

visible, and colored yellow, orange or red depending on their significance. Then the user can decide if he wants to take the risk, or if he waits for a better moment.

The exact method that determines when to trigger a warning is outlined in section 5.

4.4. Anonymity protection

Sometimes, the causes for an anonymity drop are not in the network, but in the kind of Tor usage itself. For example by connecting to uncommon ports, Tor is forced to restrict the possible exit nodes to a small subset of those. For example, there are around 1500 exit nodes in the network, but only 3 to 5 supporting port 25 (SMTP). This leads to really bad anonymity guarantees, even for non-powerful attackers. A single request of such a port advises Tor to build only circuits allowing that port for five minutes. As Tor circuits hold for 10 minutes, a single connection to an uncommon port can harm anonymity for up to 15 minutes.

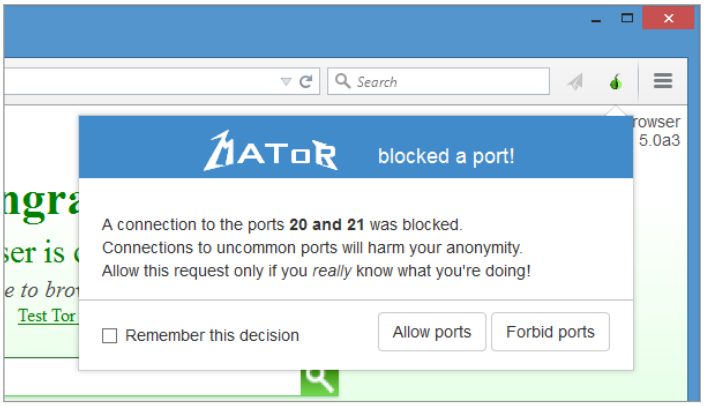


Figure 7: Warning when using FTP

The MATOR addon protects actively against this threat. If a network request occurs, that uses a destination port different from 80 (http) and 443 (https), the request is denied and a warning presented (Figure 7). Then, the user can choose if he wants to take the risk or not, since no damage is done that moment. If he chooses to allow that port, MATOR recalculates the new anonymity guarantees, that show the impact of this decision (Figure 8). The addon continues to watch the port usage, and resets anonymity to the old level, if no more circuit created with that port is in use.

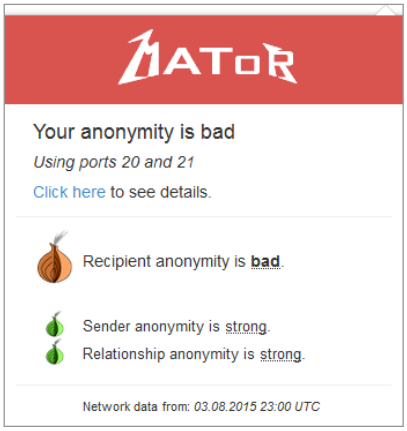


Figure 8: Anonymity with FTP

This functionality also prevents unintentional and possibly malicious connections to uncommon ports. For example, it's easy to perform network requests to any port by plain html code, any website could basically start these requests and harm anonymity. On the other hand, some sites serve downloads from FTP servers. Users clicking on a download link are not aware of the difference, and the impact it has on anonymity. The MATOR addon protects against these threats.

To prevent users from getting annoyed by these popups, the addon offers to add ports to a white- or blacklist. Whitelisted ports are always allowed (while still impacting the

anonymity value when used), blacklisted ports are blocked without further notice.

4.5. Details and statistics

For the curious users, the MATOR addon provides an overview over the anonymity values of the past (Figure 9). With these statistical graphs, on can see the development of anonymity guarantees over time. In case of a warning, these statistics might provide useful information to see why the warning was triggered. For a better overview, users can choose to see only values from a specific time range, like the last week. To generate comparable statistics, only values created by the same attacker and the default ports are considered.

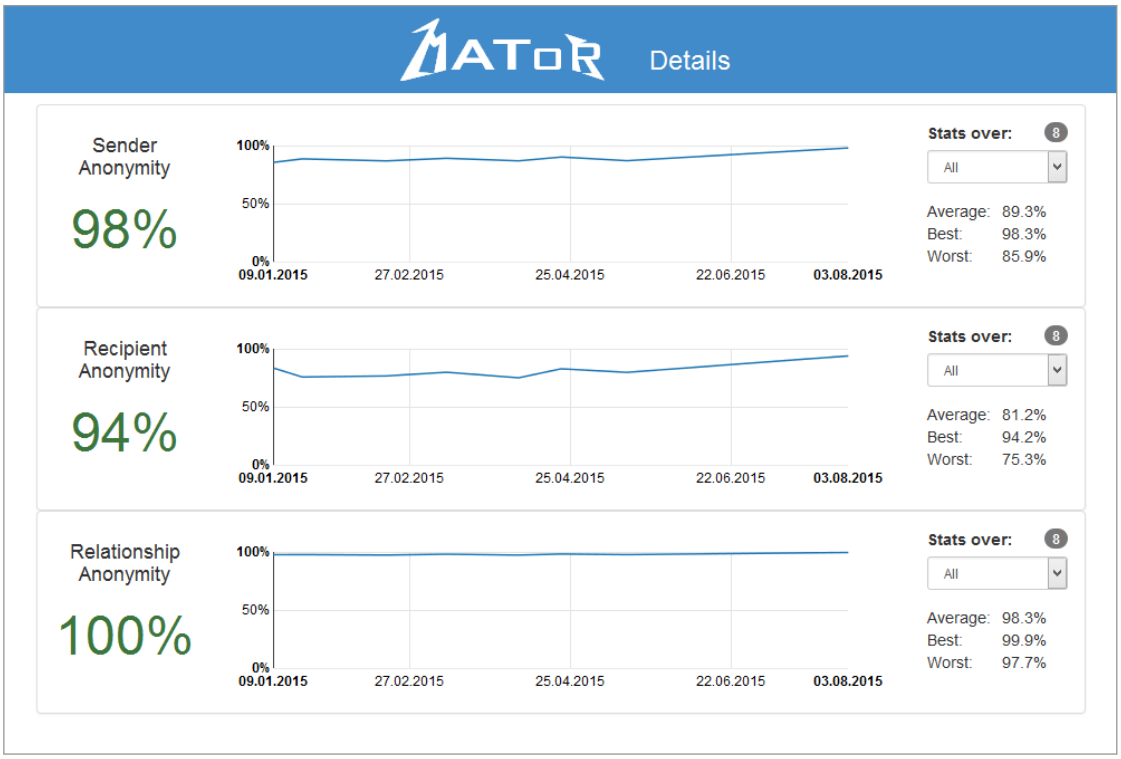


Figure 9: Details and statistics about the anonymity guarantees

4.6. Attackers

The MATOR addon provides a convenient way to select an attacker model (Figure 10). This attacker model will be used to calculate all anonymity guarantees. This way, users have the ability to "choose their enemy", depending on who or what they want to defend against. The addon implements three model classes:

- *k-of-N attacker*: The simplest attacker model. Users can select a number k of nodes. This is the default attacker model (with $k = 12$).

- *Bandwidth attacker*: Users can enter their enemy's estimated bandwidth, the default value is 10 MB/s. The bandwidth is estimated from a server's weight in the consensus.
- *Country attacker*: Users can choose a country or a combination of countries (like the "five eyes" states). Additionally, they can give this attacker a specific amount of bandwidth for servers outside of the selected countries. Default is "five eyes states" without additional bandwidth.

Of course, choosing an attacker requires some basic understanding of the system. Otherwise it happens that users select way too powerful attackers (for example by choosing a bandwidth-based attacker with more capacity than the whole network). Therefore, only attackers that have a chance of at most 95% are accepted.

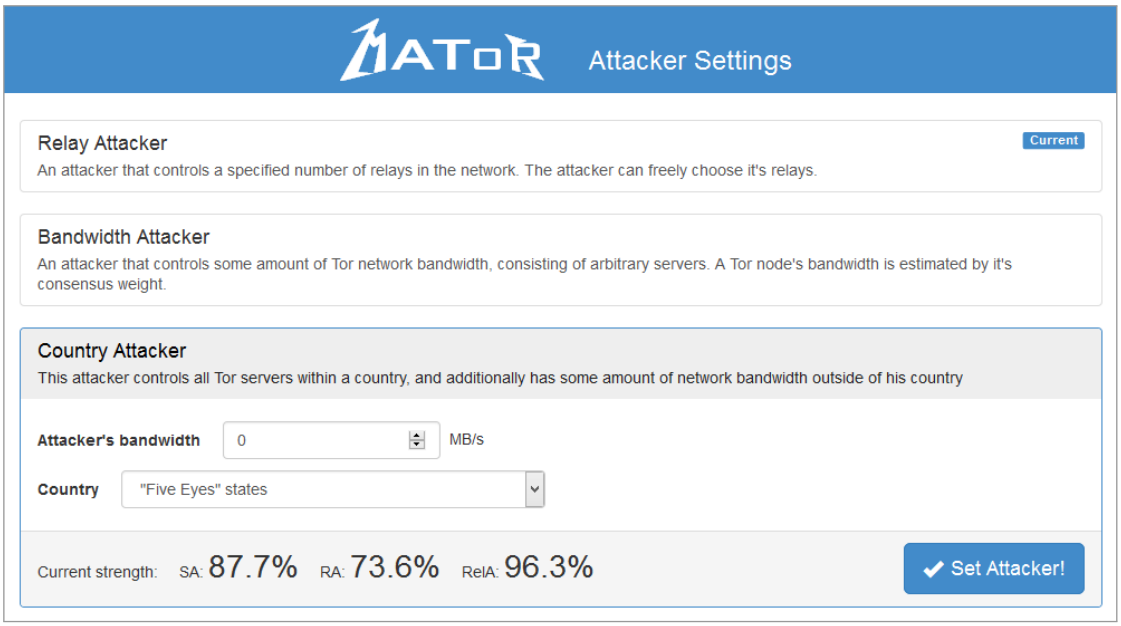


Figure 10: Attacker selection menu, shortened

4.7. Other features

For Tor Browser addons, a prime priority is that they are not risky to use. Therefore, the entire MATOR addon works offline, no outgoing network connections are made, not even through the Tor network. All data is collected locally, from the running Tor client. For attackers, it's not trivial to detect if a user is running the addon or not. However, it is possible to detect the addon if Javascript is allowed, by creating an `<img>` tag, which requests an image from the addon's source using a `resource://` url. Due to the Same-Origin-Policy, access to that image is prohibited, but using the `onload` and `onerror` callbacks, an attacker can determine if the image exists or not. This is a general issue of Firefox addons, and nothing that can be fixed in an addon.

5. Rating anonymity

As mentioned before, MATor calculates attacker's success chance in a defined game, as δ-values from 0 to 1. But most users of the MATor addon do not really understand these values, since they do not have the necessary background knowledge. As the addon should be useful for all Tor Browser users, these values are converted into the easily understandable notion of *anonymity guarantees* (see section 4.2). An anonymity guarantee is basically the user's chance to stay anonymous (the chance of an attacker to fail), which is $1 - \delta$.

As long as not stated otherwise, the addon and this work always use anonymity guarantees instead of MATor's δ-values.

5.1. Anonymity levels

Even if anonymity values are easier to understand than attacker advantage, they have two main problems.

First, users knows that 80% are better than 60%, but nobody has an absolute understanding of these values. It isn't possible to say if a value of 80% is good or not.

Second, the values are highly dependent on the selected attacker. If we know that the current sender anonymity guarantee is 80%, we can't say if that's "good" or "enough" without further knowledge. In this case, if we have selected an "5-of-n" attacker, this would be an exceptionally bad value, but if the selected attacker is "90-of-n", 80% are really good.

To solve these two problems, we do not consider absolute anonymity values, but only values compared to historical data from the same attacker. For intuitive understandability, we *rate* the guarantees and put them in one of five categories, ranging from *strong* to *dangerous*. All these categories have their own color for easy understanding: from green (good) to red (bad).

The five anonymity levels are:

Strong
Anonymity guarantees that are better than the average of historical data, are rated as "strong". They indicate that it's a good moment to use the network.

Good
Anonymity guarantees that are not better but close to the average historical guarantees, are rated as "good". They indicate that there is nothing to worry about, even if there are moments where anonymity is better.

Suspicious
If the anonymity guarantee drops below a given treshold, it is rated as "suspicious". If that's the case, the user gets a yellow warning popup. This level advises the user to use the network only if he has to, or only for less critical things.

Bad
Guarantees that are exceptionally worse, or are dropping exceptionally fast, are rated as "bad". They indicate that it's a really bad moment to use the network, and show a red warning popup to the user. Values with this level are extremely rare.

Dangerous
If the anonymity guarantees are too worse to be called bad, they will be rated as "dangerous". A user seeing dangerous guarantees should not use Tor at that time. This anonymity level also shows a red warning popup to the user. In normal Tor usage, this level should not occur. Such losses might only be caused by bigger network problems (if a big fraction of nodes is offline), or if the user's settings are really uncommon (like requesting multiple uncommon ports).

5.2. The rating algorithm

To judge anonymity values, the MATor addon includes a rating algorithm. The algorithm takes as input the current anonymity guarantees and a database of historical results, and returns one of the categories from 5.1. To rate a single value in a single setting (sender, recipient and relationship anonymity), there are three methods implemented. The resulting anonymity category for each setting is the worst of all three results. All anonymity settings are rated independently, the overall anonymity category is the worst category of these three.

- First, a **short-time comparison** is done, the anonymity guarantee values are compared to the latest available results, to detect sudden anonymity losses. The algorithm takes the five most recent results (within 24 hours), and compares each one with the current value. If the current value is at least 9% (relative) and 0.05 (absolute) lower than at a quarter of the available results, the anonymity is rated as "suspicious". If the current value is at least 12% and 0.06 lower than a third of the available results, the anonymity is rated as "bad". This comparison should detect for example ongoing DDoS attacks on Tor nodes, or changed user settings (like port usage) that pose a threat on anonymity.

  The short-time method is good at detecting sudden changes, but really depends on up-to-date results, which are not always available. Also, it can't give a measurement on how good the anonymity is on the current day.

- The addon implements a second rating method, called the **current-month comparison**. As the name suggests, it considers all results from the last 30 days, excluding the current day, and compares the current value to the average of these results. If it is above, it is rated as "strong", if it is below, but close to the average, it is rated as "good", if it is clearly below the average (12% relative and 0.032 absolute), it is rated as "suspicious". Also, the value is compared to the worst value of the last month. If it is clearly below (at least 18% and 0.04), it is rated as "bad", if the difference is even bigger (25% and 0.05), it is rated as "dangerous".

- Finally, the algorithm also does a **long-term comparison**, designed to detect slowly proceeding attacks. This method compares the current values to all available results that are older than a month. It basically works like the current month comparison, but has higher tolerance values before rating as "suspicious" or worse.

The MATor addon always uses only matching historical results, that means: only results that have been calculated using the current attacker and the default port settings (ports 80 and 443). This way, the ratings don't depend on settings from the past.

When the attacker model is changed, the addon does not have any historical information to compare against. In this situation, the addon has to *calibrate* itself. It chooses up to 15 historical consensus files from the last hours, the last month and the time before, and calculates historical anonymity guarantees for new attackers. This calibration takes around 1 to 4 minutes, until a rating can be given.

If the addon is installed for the first time, there are no historical results, and not even collected consensus files. Therefore, the addon ships with 10 consensus files from the first half of 2015. After the first calibration, they will allow the long-term comparison to work, while the other two methods have no data to work on. After the addon has been installed for some hours, the short-term method will find enough data to work on, and after a few days, the whole rating algorithm is available. This way, the addon can perform the best possible rating, even if not all needed data is available.

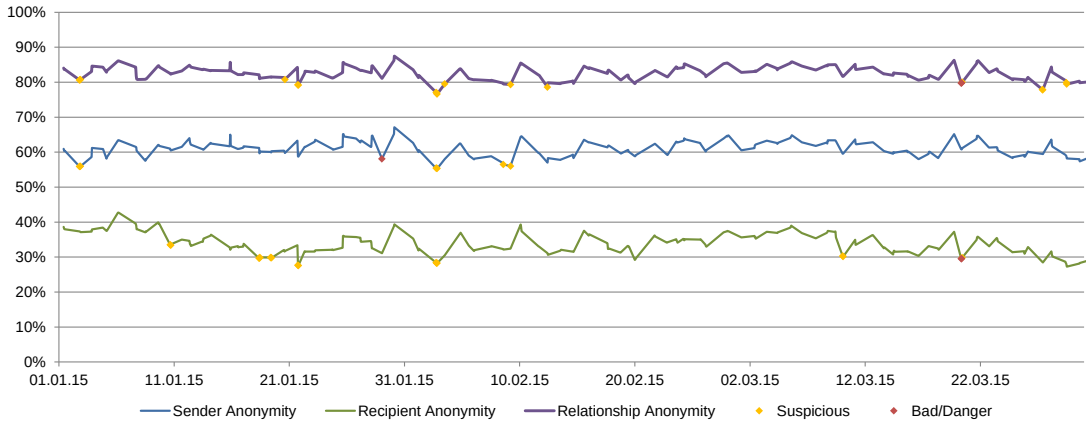
5.3. Evaluation

The rating algorithm is essential for the usability of the addon. Users should receive reliable warnings if the anonymity gets really worse, without being bothered by too many or wrong warnings. It is important to avoid wrong positives, otherwise the users will get used to warnings, and will stop taking them serious. That should clearly be avoided.

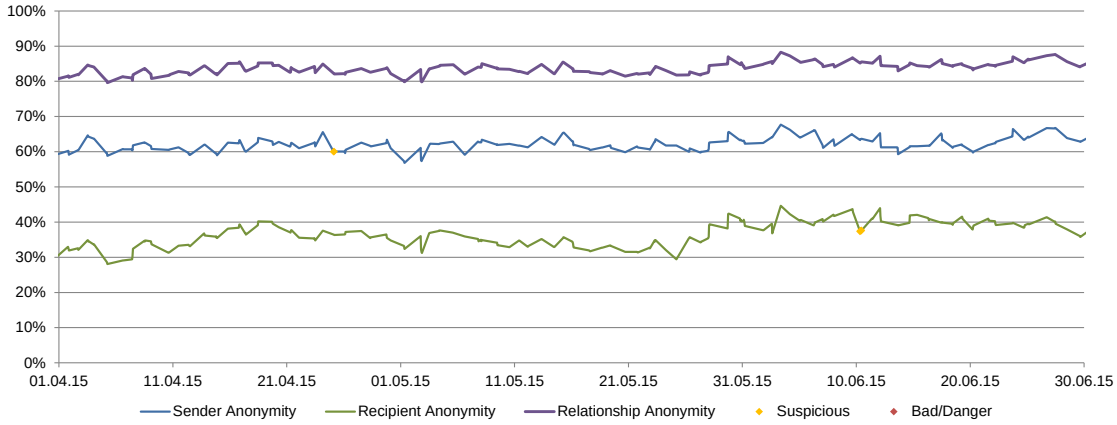
To see how good the rating works on real Tor data, I performed an evaluation. In my simulation, the Tor Browser including the MATor addon is used every day, at a random time, for up to four hours. Evaluated is the first half of 2015 (January to June). Before that time, the addon was not used regularly, but has 5 consensus files from each November and December 2014. The evaluated attackers are 5-of-n and 90-of-n. These two attackers capture different value ranges. 5-of-n results in really good anonymity guarantees of 85% - 98%, while 90-of-n is much stronger, anonymity guarantees are around 30% - 90%.

Figure 11a and 11b show the evaluated anonymity guarantees for 90-of-n. Yellow dots mark suspicious ratings, red ones mark bad values. There were no "dangerous" ratings issued in this time period.

The first part of the graph (January to March) is more interesting, as there are several warnings issued. In November and December 2014, the anonymity guarantees were better than in the first weeks of 2015. Therefore, every time the anonymity gets really low, a warning is issued, as the addon was expecting better results from the future.



(a) The anonymity guarantees in the first quarter of 2015



(b) The anonymity guarantees in the second quarter of 2015

Figure 11: Anonymity guarantees in the first half of 2015, with 90-of-n attacker

Graph 12 shows the development of the average anonymity, and the combined treshold of current-month and long-term rating. Specially the January shows how these two rating methods play together. At the beginning, the treshold is high, as the previous guarantees were better (the average line shows it). As more and more results are coming in, the treshold adjusts to the new value range. However, the long-term rating does not consider the new results. Since it is more tolerant than current-month rating, the treshold rests on a certain level, and warnings are issued if the anonymity drops deeper again. In February, the long-term rating starts considering values from January, and then the treshold slowly adjusts at the new anonymity range. Current-month and long-term rating play well together here. The current-month rating adjusts more quickly, reacting to new developments, while the long-term rating makes sure that the adjustment is not too fast.

The impact of the short-term rating can be seen on 20.03.15. As graph 13 shows,

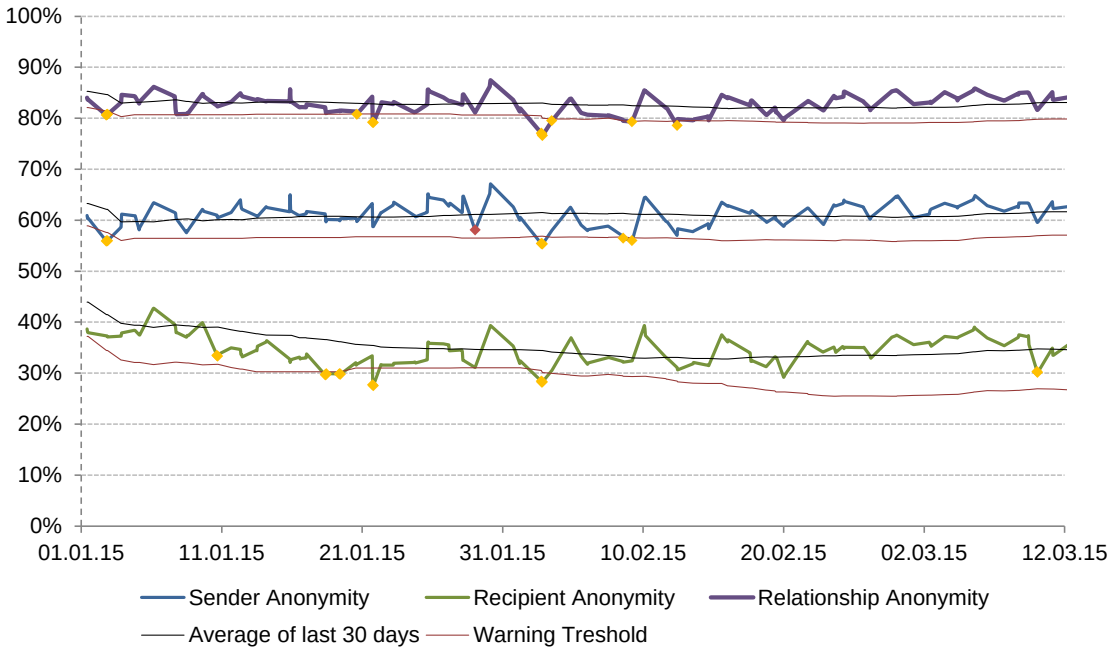


Figure 12: Anonymity values, with last month's average and warning treshold

the anonymity on 20.03 is much worser than last time, the user should know that. The recipient anonymity value dropped by 8%, to a factor of 0.8 of the previous value(38% to 30%). The recipient anonymity drops by 6.5%, here the attacker's chance rises to 150% of the old values. Both are really big changes, that are immediately rated as "bad" by the short-term rating. This graph also illustrates another challenge for the rating

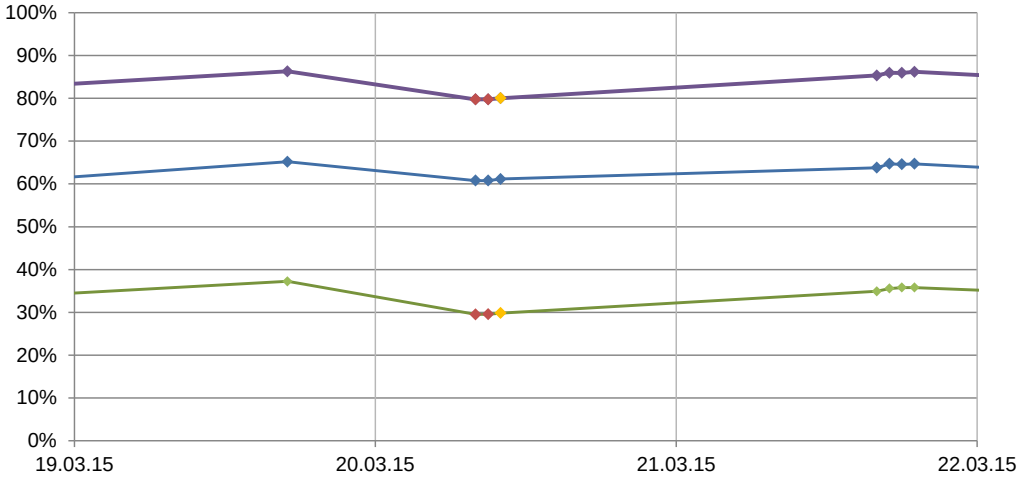


Figure 13: Anonymity dropping really fast, on March 20, 2015

algorithm. There is no reliable, or continuous database, as the Tor Browser does not run all the time. When first started on 20.03, the rating algorithm does not have data of the last hours to compare, the last values are 14 hours old.

In the second part of the graph (from April to June), the anonymity continues to improve. At that time, there's nothing to worry about, and even on minor anonymity losses, no warnings are issued (as it is still better than a month before). Only a few bigger drops trigger the short term rating. In these situations, the user can simply wait some time to use the good anonymity again.

6. The addon implementation

Basically, the MATor addon is a Firefox extension, as the Tor Browser is just a modified version of Firefox. However, the addon does not run properly in a default Firefox version, as it is missing the Tor client's information directory. The addon is mainly written in Javascript and HTML, using the Firefox Addon SDK [21]. It includes a special version of MATor, written in C and C++, packed as native binary.

Basically, the addon is split in two parts. First, there is the Addon SDK extension, which does the management, cares about interaction with Firefox and provides the user interface. The second part is the native MATor library `libmator`, which cares about calculations, optimizations and watching for new Tor data. This part is OS- and architecture-dependent. Figure 14 shows the composition of the addon in detail.

6.1. The native library

The native library is the backend of the system. It contains the MATor code for calculations, the CMATor module for management and communication with the Firefox addon, the "SimpleFileWatcher" library [29] including C bindings, and some routines to circumvent Javascript restrictions.

6.1.1. The MATor source

The base of the library is the original MATor source code. The code has been extended by additional data interfaces, that allow in-memory passing of configuration, Tor consensus data and additional node information (including information databases). Also, I added support for the consensus format of the Tor client (microdescriptor files), and implemented an "offline mode", that prevents MATor from connecting to the network. All time-consuming MATor routines are now interruption-aware and memory-safe in the case of an error. This allows addon users to cancel running calculations, otherwise the Browser would not be closable during calculation.

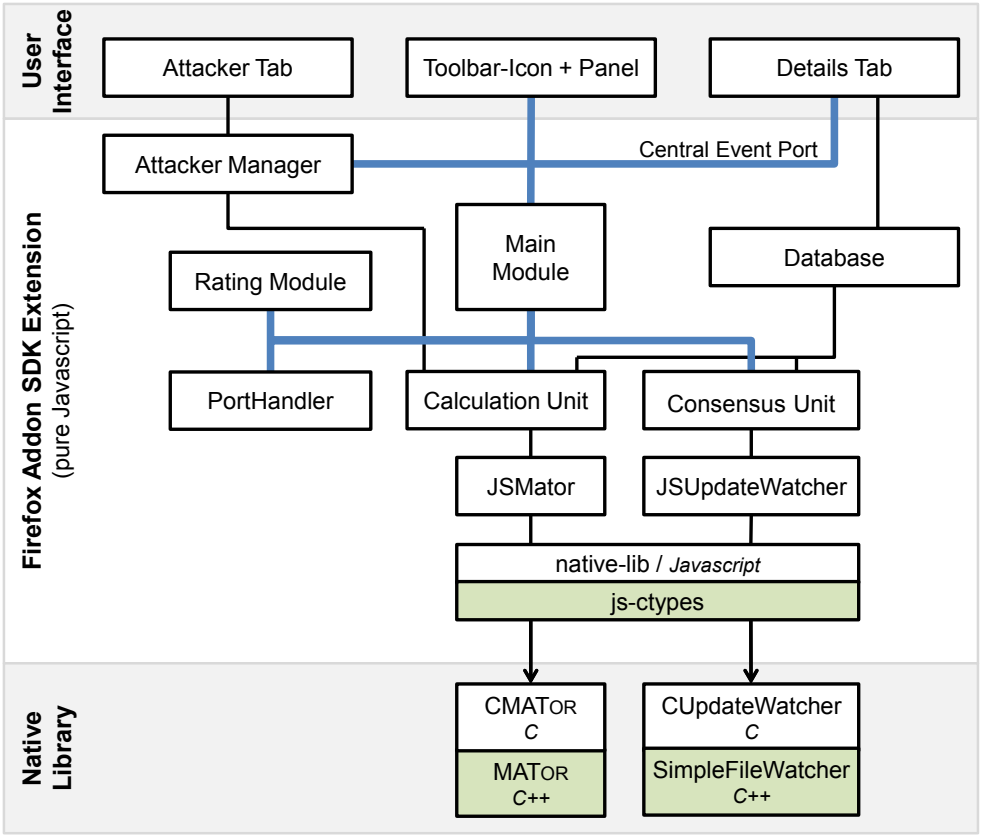


Figure 14: The addon's modules. Used libraries are green.

6.1.2. MATor optimizations

The original MATOR calculations took two to three minutes on my development machine[1], which is clearly too long for a real-time monitor. To get MATOR faster, I applied a lot of optimizations to the source code.

The hardest part of MATOR calculations was the consensus preparation phase and relationship anonymity calculation, while sender and recipient anonymity had already good approximations. I focused on the preparation phase, which has a runtime cubic in the size of the network. Careful transformations on this code nearly doubled the speed of some phases, while not altering functionality at all. Next, I parallelized all bigger parts of the code, using a threadpool and a workqueue. This gives another performance gain, especially on modern processors, without any accuracy losses.

For the relationship anonymity calculation, I slightly changed the algorithm of the underlying heuristic. My optimized version requires space only linear in the size of the network times the accuracy, while the original required quadratic. Next, I implemented the algorithm a level closer to the hardware, using pointers and pointer arithmetic instead

[1]Measured on an Intel Q6600, 4×2.4GHz, 8GB memory, both Windows 8.1 and Linux Mint

of higher-level C++ classes. This was not only faster, it also allowed the compiler to optimize the code even more (enabling compiler optimizations doubled the heuristic's speed again). Last, I added an external switch for the accuracy of the algorithm. Slightly lowering this accuracy lead to acceptable times for the relationship anonymity algorithm, even if complicated attackers have been selected.

Even with these code optimizations, MATor was still too slow for a background usage in the addon. To circumvent this problem, the MATor addon uses a trick: Instead of calculating anonymity guarantees for the whole consensus, the guarantees are only calculated on the important part of the Tor network. The impact of a node depends on it's consensus weight. Nodes with really low weights are almost never chosen, and do not really affect the calculation. It turned out that a lot of the Tor nodes are almost never used, since they simply are too small. Small nodes can only get some relevance if they are exit nodes for uncommon ports.

An evaluation of different weight thresholds showed how good this method works. In the addon, every non-exit node with a weight less than 2500 is dropped (corresponds to around 500 kb/s), also every exit node with a weight below 100 (around 20kb/s). From a consensus containing $\sim$ 6500 nodes, around 3500 nodes are dropped, which corresponds to 7% - 10% of the total bandwidth, and 0.2% of the total exit bandwidth. In my evaluation, this lead to anonymity guarantee changes of around 0.1% (absolute), even less for recipient anonymity. The addon shows at most one digit after the period, so this difference is totally fine. In contrast, a calculation now takes 10 to 15 seconds on my development machine, and even less on a modern laptop.

6.1.3. CMator

CMator is the wrapping component for the original MATor code. It is the bridge between native and Javascript code, it provides a plain C interface, which can be accessed from the Firefox addon. The module collects data given by the addon, including special node information like the GeoIP database, and creates MATor instances as needed. The module also cares about threading, interrupting of calculations and error handling. Also, all memory management is done here.

6.1.4. Other parts

The native library also contains a modified version of the *SimpleFileWatcher* library [29]. It provides a convenient wrapper around operating-system-dependent functions, that notify if a specific file has been changed. The library has been slightly modified by me, since the original version contained severe bugs. To allow the Firefox addon to use this method, the *UpdateWatcher* component provides C bindings to the library.

Finally, the library provides two methods to convert pointers from and to a serializable Javascript type. This is needed to work around restrictions in Javascript's threading model.

6.2. From C to Javascript

While the native library is written in C and C++, modern Firefox addons are built using web technologies, e.g. Javascript, HTML and CSS. These language follow different paradigms. While Javascript is a managed language, which is interpreted (or just-in-time compiled) by the Browser, C and C++ are completely unmanaged and compiled by the developer. Also, Javascript programs run in a sandbox, which they can only escape by provided modules.

Luckily, Firefox provides a Javascript module called js-ctypes [22], which can be used to load dynamic libraries in the operating system's format, and create Javascript bindings for the contained C methods. Also, it provides Javascript wrappers for the most basic C types, including conversions (numbers, floating numbers, `char*` strings, `void` and pointers). Using js-ctypes, the addon loads the native library from it's extension directory, and accesses the methods inside.

However, js-ctypes has some drawbacks. First, it has no (working) method to manage the memory. In contrast to C, Javascript has a garbage collector, which removes inaccessible objects. This makes it difficult to pass objects from Javascript to C, as the Javascript garbage collector might free the objects, while C code still maintains references to it. On the other hand, the garbage collector does not free C objects when removing their reference. I solved these problems by moving the whole memory management to the library. As far as possible, all objects are created by the C code, or copied as soon as they reach it. Also, the C part maintains a list of all objects, and frees them only if no Javascript reference can exist any more, or if the library is going to be unloaded. If I have to pass uncopyable objects to C (like function callbacks), a reference to these callbacks is also stored in the Javascript world, preventing the garbage collector from freeing it.

The second drawback of js-ctypes is it's limited interoperability with Javascript sandboxes. No ctypes object can be passed from one sandbox to another. When calculating anonymity guarantees, this has to be done in the background. C++ threads are not usable here, since they can't notify the addon when they are done (because of Javascript's missing theading features). To solve this, the background thread is a Javascript "Web Worker" (more precise: a `ChromeWorker`). Javascript code in background workers runs in another isolated sandbox, which implies that no C objects can be passed there. This can only be circumvented in a fairly ugly way: The background worker has to load the library a second time and bind the needed wrapper methods again. Then it receives the needed calculation data (a pointer converted to a string), and can start calculating.

Using all these techniques, I rebuilt the whole CMator and UpdateWatcher structure as clean Javascript objects, having roughly the same API as their C++ equivalents. Finally, the addon's code can use the `JSMator` and `JSUpdateWatcher` without caring about everything behind.

```
// in consensus unit
port.emit("load-consensus", {consensus: "...", microdescFile: "..."});

// in calculation unit
port.on("load-consensus", function (consensus) {
    // load consensus.file
    port.emit("start-main-calculation");
});

// in user interface
port.on("load-consensus", function (consensus) {
    // display date and time of the new consensus
});
```

Listing 1: Sample usage of the event port, showing the internal communication

6.3. The Central Event Port

The addon's Javascript code uses event-based programming. An `EventPort` connects all important components together (see Figure 14). Each connected module can send messages over the port, which can include attached data. Other modules can listen on such messages, and react to them. An example event is `load-consensus`, which is issued when a new consensus file has been downloaded, including information about the consensus file. Other examples are `calculation-started`, `ports-changed` (including list of ports), `main-rated-results` (including results and ratings) and `open-details`. Listing 1 shows an example usage of the event port.

The event port allows easy extensibility of the system, since all components talk a common language. New modules only need access to the event port, and can be notified about every interesting event that might occur. There is no need to add code to foreign components, if the new module needs for example the current anonymity guarantees.

6.4. The calculation modules

The calculation of the current anonymity guarantees is mainly done by the calculation unit, which collects all needed input, and progresses it using JSMator. Each calculated result is stored in a database. Whenever a new calculation is started, the calculation unit first checks if there is already an result in the database, to prevent unnecessary calculations. The result database is also used to display the statistics.

When Tor receives a new consensus file, the consensus unit is triggered. This unit loads the new consensus file, compresses it and stores it into the database, to allow further calculations on it. Next, it passes the new consensus to the calculation unit, and triggers a recalculation of the anonymity guarantees using the event port. This unit also located microdescriptors and GeoIP databases, which are sent over the event port.

After new anonymity guarantees have been calculated, the rating module classifies them. It looks up historic guarantees from the database, and applies the algorithms from section 5. If there are not enough results in the database, the rater triggers calculations

on stored consensus files from the database, to get enough data to work on. Finally, the ratings and the guarantees are sent to all frontend modules, via the event port.

6.5. The Frontend

In Firefox addons, the user interface is mainly implemented using the web technologies HTML and CSS. The interface is split into single components, which act independent of each other. Each component is connected to the central event port, where they get all needed information from. User actions are passed on the event port, which allows the remaining addon modules to react to them.

The backend part of the details tab additionally has access to the database, which is used to fetch matching historic results to display the graphs. The backend part of the attacker-selection tab has additional access to the calculation unit, to calculate anonymity guarantee previews for new attacker models.

6.6. Port blocking

As an additional security feature, the MATor addon blocks all connections to uncommon ports, which means: everything which is not local, to port 80 (http) or 443 (https) is blocked by default. This is justified by the anonymity impact that uncommon ports have on recipient anonymity. Technically, this is done via a Javascript implementation of `nsIContentPolicy`, which is a C++ interface for older addons (XPCOM addons).

The porthandler module includes it's own user interface module, to warn users about the usage of uncommon ports. However, users can select to allow access to a specific port. In this case, the porthandler module notifies all other components, sending them the new list of used ports. A new calculation is triggered, in which the calculation unit takes the new ports into account. The porthandler module also checks how long an uncommon port might influence Tor's circuit generation. When no influenced circuit can exist any more, the module notifies all other components again.

6.7. Unit testing

The important parts of the addon are covered by automatic unit tests. A first test suite is testing most functions of the native library, including the SimpleFileWatcher library and MATor itself. This test suite is powered by Boost's Unit Test Framework [7].

A second test suite is testing the most important parts of the Firefox Addon code. It is written using the `cfx test` command of the Mozilla Addon SDK.

6.8. Assembling the addon

Assembling the addon is a three-step progress: As first step, the native code is compiled and bundled together to the native library. The compilation happens on a Linux host, targeting Windows and MacOS by cross compilers (mingw-w64 for Windows, and OSXCross [25] for MacOS). The built libraries are stand-alone, only linking against the operating system's interfaces. All third-party libraries (like Boost, SQLite and GLPK)

are statically linked, which requires manual compilation for them. This way, the resulting library runs on any system without any further dependencies.

As second step, the addon code is bundled together by the grunt task runner. All `less` files are processed to `css` stylesheets, all external libraries are compressed, combined and moved into the right directory, and finally the native libraries from the first step are copied to the data directory.

As last step, the `cfx` tool from the Mozilla Addon SDK packs all these files together in an `xpi` file, which is ready to be installed in the Tor Browser.

7. Conclusion and future work

The MATor addon provides easy access to reliable anonymity guarantees, even for laymen. The anonymity guarantees are based on the current network status from the running Tor client, a selected threat scenario and a user's preferences. With little overhead, the addon runs silently in the Tor Browser. It warns if anonymity gets bad, or a user's actions might harm his anonymity. With it's simple usage, the MATor addon brings current research on Tor's anonymity to Tor's users.

Tor users that rely on Tor's anonymity (like whistleblowers) can now install the addon, and use Tor with less fear, while anonymity is good enough for them. If the Tor network becomes more vulnerable to their selected attacker, the anonymity gets bad and they will be warned. Now they can stop using it and avoid the (higher) risk of being catched. After waiting a few hours, they can check again how the anonymity developed, and decide if it's safe enough to use Tor. Now, these users have all information they need to make a decision about using Tor, and don't have to rely blindly on it.

Future work. There are many possibilities to build on this work. When getting warned by the MATor addon, it is not always clear *why* the anonymity is bad at all. For interested users, it would be nice to show them the main causes of an anonymity loss (like the appearing of a new, fast server). This will help users to make an informed decision whether or not to use Tor at this moment.

Next, it would be interesting to include more attacker types into the addon. Currently, the MATor people are investigating attackers that control several network segments, like a submarine cable, an Internet exchange point or a communication satellite. In addition to the general knowledge about Tor's anonymity, these attacker types are highly interesting in the live monitor. Maybe even concrete circuits become assessible, by comparing their route with the selected attacker's segments.

On a technical level, the MATor addon might be improved by moving away from Javascript code. Languages like Typescript [9] provide much more features, like static typechecking, which will speed up the development and improve quality of the product. A conversion to Typescript will require some time, but might pay off in the future.

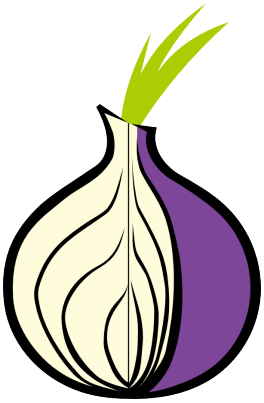
Since not all Tor users are using the Tor Browser, it would be nice to port the live monitor to other targets. For mobile devices, there is *Orbot*, the Android equivalent of the Tor Browser. An Orbot extension would be the way to run MATor on mobile

devices. However, it's questionable if these mobile devices have the computational power to run MATor in acceptable times, with acceptable battery consumption. Before implementing the addon for mobile devices, the performance of the MATor framework must be dramatically improved.

People that need the highest anonymity protection available often use *Tails*, a live Linux distribution with integrated Tor client. An integration into the Tails live disk would bring MATor to even more users.

Beside typical users, there are also servers running hidden services in the Tor network. For the server administrators, a MATor extension would be interesting. This extension might for example take the server offline if anonymity gets too bad, or cut of specific services depending on the current degree of anonymity.

As outlined, the MATor addon is only the first step in bringing anonymity measurement tools to Tor's users. More and better tools for user-friendly anonymity monitors are yet to be developed.



References

[1] *about-what*. URL: https://www.npmjs.com/package/about-what (visited on 08/10/2015).

[2] *AngularJS — Superheroic JavaScript MVW Framework*. URL: https://angularjs.org/ (visited on 08/10/2015).

[3] *Angularjs Nvd3 Directives*. URL: https://angularjs-nvd3-directives.github.io/angularjs-nvd3-directives/ (visited on 08/10/2015).

[4] *AngularStrap*. URL: https://mgcrea.github.io/angular-strap/ (visited on 08/10/2015).

[5] Michael Backes et al. "AnoA: A Framework For Analyzing Anonymous Communication Protocols". In: *Proceedings of the of the 26th IEEE Computer Security Foundations Symposium (CSF)*. IEEE, 2013, pp. 163–178.

[6] Michael Backes et al. "(Nothing else) MATor(s): Monitoring the Anonymity of Tor's Path Selection". In: *Proceedings of the 21st ACM Conference on Computer and Communications Security (CCS)*. ACM, 2014, pp. 513–524.

[7] Boost.org. *Boost C++ Libraries*. URL: http://www.boost.org/ (visited on 08/08/2015).

[8] Mike Bostock. *D3.js - Data-Driven Documents*. URL: http://d3js.org/ (visited on 08/10/2015).

[9] Microsoft Corp. *Welcome to TypeScript*. URL: http://www.typescriptlang.org/ (visited on 08/10/2015).

[10] Yossi Gilad and Amir Herzberg. "Spying in the Dark: TCP and Tor Traffic Analysis". In: *Privacy Enhancing Technologies*. Ed. by Simone Fischer-Hübner and Matthew Wright. Vol. 7384. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2012, pp. 100–119.

[11] *GLPK - GNU Project*. URL: https://www.gnu.org/software/glpk/ (visited on 08/10/2015).

[12] Nicholas Hopper, Eugene Y. Vasserman, and Eric Chan-tin. "How much anonymity does network latency leak". In: *In CCS '07: Proceedings of the 14th ACM conference on Computer and communications security. ACM*. 2007.

[13] Amir Houmansadr and Nikita Borisov. "SWIRL: A scalable watermark to detect correlated network flows". In: *In Network and Distributed System Security Symposium. Internet Society*. 2011.

[14] The Tor Project Inc. *Tor Browser*. 2015. URL: https://www.torproject.org/projects/torbrowser.html.en (visited on 08/07/2015).

[15] The Tor Project Inc. *Tor Metrics — Relays with Exit, Fast, Guard, Stable, and HSDir flags*. 2015. URL: https://metrics.torproject.org/relayflags.html (visited on 08/08/2015).

[16] The Tor Project Inc. *Tor Project: Anonymity Online*. 2015. URL: https://www.torproject.org/ (visited on 08/08/2015).

[17] The Tor Project Inc. *Tor Project: FAQ*. 2015. URL: https://www.torproject.org/docs/faq.html.en (visited on 08/07/2015).

[18] *jQuery*. URL: https://jquery.com/ (visited on 08/10/2015).

[19] Andre Meister. *Homepageüberwachung: Bundeskriminalamt hat mehr als 150 Fahndungsseiten überwacht*. 2012. URL: https://netzpolitik.org/2012/homepage-uberwachung-bundeskriminalamt-hat-mehr-als-150-fahndungsseiten-uberwacht/ (visited on 08/08/2015).

[20] MozillaWiki. *Fingerprinting - MozillaWiki*. 2015. URL: https://wiki.mozilla.org/Fingerprinting (visited on 08/10/2015).

[21] Mozilla Developer Network. *Add-on SDK*. 2015. URL: https://developer.mozilla.org/en/Add-ons/SDK (visited on 08/08/2015).

[22] Mozilla Developer Network. *js-ctypes*. URL: https://developer.mozilla.org/en-US/docs/Mozilla/js-ctypes (visited on 08/08/2015).

[23] *NVD3*. URL: http://nvd3.org/ (visited on 08/10/2015).

[24] Mark Otto and Bootstrap contributors. *Bootstrap - The world's most popular mobile-first and responsive front-end framework*. URL: http://getbootstrap.com/ (visited on 08/10/2015).

[25] Thomas Pöchtrager. *OSXCross*. URL: https://github.com/tpoechtrager/osxcross (visited on 08/08/2015).

[26] *SQLite Home Page*. URL: https://www.sqlite.org/ (visited on 08/10/2015).

[27] *TheRusskiy/ng-slide-down*. URL: https://github.com/TheRusskiy/ng-slide-down (visited on 08/10/2015).

[28] *Underscore.js*. URL: http://underscorejs.org/ (visited on 08/10/2015).

[29] James Wynn. *simplefilewatcher - Simple, cross platform, object-oriented, file watcher and notifier library*. URL: https://code.google.com/p/simplefilewatcher/ (visited on 08/08/2015).

A. Used tools and libraries

The native part includes this libraries:

- **MATor** [6], developed by the "Information Security and Cryptography" Group at Saarland University
- **Boost** library [7], for threading and unit tests
- **SQLite** [26] and **GLPK** [11] *(GNU Linear Programming Kit)*, used by MATor
- **SimpleFileWatcher** [29], used to get notified on consensus changes

The native part is built using these tools:

- **GNU Make** is managing the whole build process
- **GNU GCC** and **G++** compile the code for linux
- **Mingw-w64** is the crosscompiler used to target windows systems
- **OSXCross** [25] is used to get an OSX-targeting gcc-based crosscompiler.

The Firefox addon includes this libraries:

- **Firefox Addon SDK** [21], to interact with Firefox
- **Underscore.js** [28], provides useful functions in front- and backend
- **Bootstrap** [24], as frontend design library
- **JQuery** [18], as lightweight frontend framework
- **AngularJS** [2], as framework for details and attacker view
- **d3** [8] and **nvd3** [23] as graph libraries
- **Angular-strap** [4] and **angularjs-nvd3-directives** [3], AngularJS-bindings for bootstrap and the nvd3 library
- **ng-slide-down** [27], widget for the attacker tab
- **AboutWhat** [1], a wrapper to register `about:*` urls

The whole addon's built progress is handled by these tools:

- **NodeJS** and it's package manager **npm** to run the build process
- **Bower** for front-end dependency management
- **Grunt** including various plugins, as task runner for everything. To see the full plugin list, take a look at the project's `package.json`.
- The **cfx** tool from the Addon SDK, to build xpi files
- **Less**, as CSS preprocessor
- **Extension Autoinstaller**, which allows fast development

Credits go to their respective authors.