Bento: **Safely Bringing Network Function Virtualization to Tor**

Michael Reininger
University of Maryland

Arushi Arora
Purdue University

Stephen Herwig
University of Maryland

Nicholas Francino
University of Maryland

Jayson Hurst
University of Maryland

Christina Garman
Purdue University

Dave Levin
University of Maryland

ABSTRACT

Tor is a powerful and important tool for providing anonymity and censorship resistance to users around the world. Yet it is surprisingly difficult to deploy new services in Tor—it is largely relegated to proxies and hidden services—or to nimbly react to new forms of attack. Conversely, "non-anonymous" Internet services are thriving like never before because of recent advances in programmable networks, such as Network Function Virtualization (NFV) which provides programmable in-network middleboxes.

This paper seeks to close this gap by introducing programmable middleboxes into the Tor network. In this architecture, users can install and run sophisticated "functions" on willing Tor routers. We demonstrate a wide range of functions that improve anonymity, resilience to attack, performance of hidden services, and more. We present the design and implementation of an architecture, Bento, that protects middlebox nodes from the functions they run—and protects the functions from the middleboxes they run on.

Bento does not require modifications to Tor, and we evaluate it by running it on the live Tor network. We show that, with just a few lines of Python, we can significantly extend the capabilities of Tor to meet users' anonymity needs and nimbly react to new threats.

CCS CONCEPTS

• **Networks** → **Middle boxes / network appliances**; **Network privacy and anonymity**; *Programmable networks.*

KEYWORDS

Tor, Network Function Virtualization (NFV), Intel SGX

ACM Reference Format:
Michael Reininger, Arushi Arora, Stephen Herwig, Nicholas Francino, Jayson Hurst, Christina Garman, and Dave Levin. 2021. Bento: Safely Bringing Network Function Virtualization to Tor. In *ACM SIGCOMM 2021 Conference (SIGCOMM '21), August 23–28, 2021, Virtual Event, USA.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3452296.3472919

SIGCOMM '21, August 23–28, 2021, Virtual Event, USA

ACM ISBN 978-1-4503-8383-7/21/08…$15.00
https://doi.org/10.1145/3452296.3472919

1 INTRODUCTION

Anonymity systems are critical in achieving free, open communication on today's Internet. Tor [24] in particular has become a staple in resisting censorship and allowing journalists to safely communicate with their sources [65].

Das et al. [22] described the fundamental trade-offs that anonymity systems must make as an "anonymity trilemma": no one system can simultaneously achieve strong anonymity, low latency, and high bandwidth. Tor trades off strong anonymity in favor of greater performance; DCNets [19] trades off performance for strong anonymity. The anonymity trilemma tells us that each anonymity system must cement its place in this design space, and users in turn must choose the system that suits their anonymity goals.

In this paper, we show that there may be another way to overcome this trilemma. While all three properties may not be simultaneously achievable for all users, we argue that *a more programmable anonymity network* can let users choose the precise set of trade-offs they want, when they want them.

We present Bento, a novel architecture that augments Tor by allowing relays to act as user-programmable "middleboxes." Bento allows clients to write sophisticated middlebox "functions" in a high-level language (Python, in our implementation) and run them on willing Tor relays. For example, we present a basic network function that adds bidirectional cover traffic to a circuit, thereby temporarily achieving stronger anonymity for a subset of the users at the cost of increased bandwidth consumption.

Bento is inspired by recent impressive innovations in network function virtualization (NFV). NFV provides programmable in-network middleboxes [7, 8, 16, 38, 43, 47, 72, 74] that can be used to construct more robust, scalable and resilient network services. Bento extends prior work in NFV by demonstrating that it is possible to *safely* deploy functions on middleboxes in adversarial settings. More specifically, we operate within a threat model that prior NFV work has not explored: one in which the client and middlebox (Tor relay) are mutually distrusting. By using recent advances in trusted execution environments [34], Bento ensures that the client is protected from the middlebox, and the middlebox is protected from the client.

We demonstrate a wide diversity of functions that significantly improve various aspects of Tor, including: (1) `Browse`: a function

that offloads a client's web browser to avoid website fingerprinting attacks, (2) `LoadBalance`: a function that automatically scales hidden service replicas up and down to handle varying load, and (3) `Dropbox`: a function that allows Tor to be used as an anonymous file store.

Bento safely extends Tor without requiring modifications. Rather, it runs on top of Tor; users terminate circuits at Bento middleboxes to deploy and execute functions. As a result, it is incrementally and immediately deployable—to this end, we have made our code publicly available at `https://bento.cs.umd.edu`.

Contributions We make the following contributions:

- We introduce the first architecture to safely bring the power of NFV to anonymity networks like Tor. Critical to many of the security guarantees of our architecture are recent developments in trusted execution environments.
- We identify, design, and implement critical components that are necessary to ensure expressiveness and safety of an anonymous middlebox architecture. Although we focus on Tor, many of these components are broadly applicable.
- We present middlebox functions that solve a wide diversity of problems that have long plagued the Tor network, including website fingerprinting defenses and more robust hidden services.
- We evaluate our prototype architecture and functions on the live Tor network and show that it extends Tor's capabilities and defenses while adding nominal performance overhead.

Roadmap We present background and related work in §2. We then present Bento's overview (§3), goals (§4) and design (§5). We analyze Bento's security properties in §6. We present two Bento functions in depth—Browser in §7 and a hidden service load balancer in §8—and then briefly describe a wider range of functions in §9. We discuss ethical concerns in §10. We summarize both the current limitations of Bento and avenues for future work in §11 and conclude in §12.

2 BACKGROUND AND RELATED WORK

In this section, we provide a broad overview of Tor and a description of our threat model. We also review related work on programmable middleboxes and on extending Tor's features, ultimately showing that there is a surprisingly large gap between the two. One of the goals of this paper is to bridge this gap by *securely* bringing NFV to anonymity systems.

2.1 Tor Background

Tor [24] is a peer-to-peer overlay routing system that achieves a particular type of anonymity known as *unlinkability*: an adversary can identify at most one of a source/destination pair, but not both. Tor achieves unlinkable communication by routing traffic through a *circuit*: a sequence of overlay hosts known as Tor relays. There are typically three relays in a circuit: an entry node (who communicates with the source), a middle node, and an exit node (who communicates with the destination). The source node is responsible for choosing which Tor relays to include in a circuit, and for constructing the circuit.

Proxied communication To anonymously communicate with a server that is *not* in the Tor network, a Tor client creates a circuit to an exit node, and then instructs the exit node to open a traditional network connection to the desired destination. Tor relays specify in their *exit node policy* which hosts and which ports they are willing to connect to (and whether they prefer not to be exit nodes at all).

Hidden services In addition to connecting to external servers, Tor supports *hidden services*[1] [50, 78], which allow users to host services anonymously. Briefly, hidden services operate as follows: To host a hidden service, one chooses a pseudonymous identifier and a set I of Tor relays to serve as *introduction points*. The hidden service creates Tor circuits to each $i \in I$ and (anonymously) publishes the mapping between its identifier and I. To connect to the hidden service, a client chooses a Tor relay r to serve as a *rendezvous point*, and creates a Tor circuit to it. The client then chooses an $i \in I$, creates a circuit to it, and requests that it forward r (and some additional information) to the hidden service. In turn, the hidden service creates its own circuit to r. After this process, r serves as a bridge between the client and hidden service circuits, providing connectivity between the two.

2.2 Threat model

Bento runs on top of the existing Tor network, and as a result we adopt the same network-level threat model. This can vary by user and application, but a common assumption is that of a powerful routing-capable adversary [67], such as a nation-state. Such an adversary often controls a large network—and can even influence nearby routes to go through its network—but cannot have a *global* view of Internet traffic. Adversaries can also actively participate in the Tor network [3].

In addition to these routing-capable, network-based adversaries, our architecture requires us to consider the threats that can arise from an altogether new mode of interaction: loading and running code on other users' machines. We assume that users naturally have physical access to their machines, and can thus introspect on running processes. However, we also assume that some Bento middleboxes will have secure, trusted execution environments (TEEs), such as Intel SGX. We explicitly assume that these environments are safe; that is, for any code or data being executed or stored inside of a secure enclave, we assume that the attacker cannot introspect on either, despite having physical access to the machine. This assumption has been drawn into question by recent attack discoveries [81], but we do not believe these vulnerabilities are fundamental to TEEs (and many have been patched [68]). Nonetheless, we note that we are not strictly bound to SGX; Bento uses conclaves [34] ("containers of enclaves"), a system designed to work with any TEE with similar properties to SGX.

Attacks Tor has been targeted by both academics and real-world attackers [3, 10, 17, 27, 30, 33, 36, 41, 46, 50, 55, 56, 61, 69, 77, 86]. To show some of the potential benefits of Bento, we consider two broad classes:

First, *deanonymization attacks* [3, 15, 27, 36, 56] seek to infer the two endpoints of a Tor circuit through passive or active traffic analysis. Routing-capable adversaries are very well-suited for these kinds of attacks, as they can influence traffic on the entry leg and

[1] These are sometimes also referred to as "onion services"; Tor developers use the terms interchangeably [58].

the exit leg to go through networks they control—at that point, they can perform straightforward traffic correlation attacks [41].

Second, *fingerprinting attacks* [12, 13, 50, 82–84] observe only the traffic from the source, and use deterministic traffic patterns from web servers to act as fingerprints. Typical defenses involve reordering or batching requests and sending junk control packets to make websites appear indistinguishable from traffic patterns alone.

We describe state of the art website fingerprinting attacks in §7 and deanonymization attacks in §9.1.

2.3 Programmable Middleboxes

Middleboxes are network devices that sit on the traffic path (often between two routers or switches) and perform processing on packets as they traverse the network. Historically, middleboxes were monolithic (a single box served a single purpose, such as a firewall or load balancer) [28, 32, 39, 44, 70, 85]. Moreover, unlike typical computers, traditional middleboxes were not re-programmable: often, simply getting a new version required getting an entire new physical device.

Recent innovations in *network function virtualization* (NFV) [7, 8, 16, 38, 39, 43, 47, 72, 74, 85] allow network operators to instantiate middleboxes in virtual machines and place those VMs at arbitrary locations in the network [29]. Some approaches have explored how to make NFV more easily programmable by means of constructing modular, composable *functions* that can be deployed onto middleboxes [8].

Many network services today depend on a variety of middleboxes—including firewalls, load balancers, traffic shapers, or intrusion detection systems—and increasingly many are relying on programmable middleboxes with NFV. As a result, there is a significant gap between the capabilities of the "non-anonymous" Internet and the Tor anonymity network. Bento seeks, in effect, to "modernize Tor" by incorporating programmable middlebox functionality. We demonstrate that this will make it feasible to deploy a far more sophisticated set of anonymous services than is possible today.

2.4 Extensions to Tor

The closest piece of related work to our programmable middleboxes is FAN (Flexible Anonymous Network) [66]. FAN seeks to make the Tor protocol itself more programmable, allowing for custom Internet privacy and lightweight updates through the use of Protocol Plugins, which are pieces of code that are merged into the Tor codebase and executed inside a userland virtual machine. In contrast, Bento sits atop Tor, and as a result, is complementary to FAN.

Prior work has also looked to improve the security and privacy of Tor using TEEs. SGX-Tor [48] combines SGX and Tor, reducing the Tor attacker down to a network-level adversary with no insight into the internal state of Tor components. Large parts of the Tor code and data are placed into an enclave to both protect sensitive data and leverage the correctness of execution and integrity guarantees of SGX. This prevents a number of well-known attacks, including low resource attacks to demultiplex circuits, and also allows nodes to protect the list of relays used. Bento expands on this considerably, by allowing users to safely deploy *new*, tailored functions (not just the existing Tor codebase).

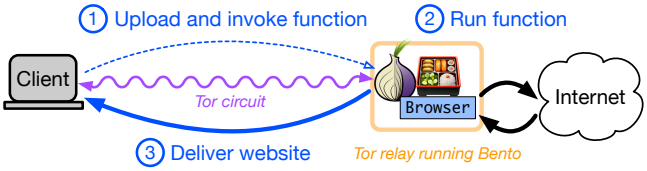

Figure 1: Overview of installing, and executing a `Browser` "function" that runs on an exit node, downloads a given URL, and delivers it, padded to some threshold number of bytes. To an attacker sniffing the client's link, it appears the client uploads a small amount and then downloads a large amount.

3 OVERVIEW OF BENTO

Before describing its goals (§4) and design (§5), we first present a high-level overview of how Bento enables users to extend Tor with programmable functions.

In Bento, a client can *offload* processing that would have happened on their own machine to *another* node in the Tor network altogether. We depict this in Figure 1, and describe each step with a motivating example.

Motivating Example A user, Alice, wishes to anonymously browse a website over Tor, but fears that an adversary has the ability to observe traffic entering and leaving her machine. Such an adversary could launch a website fingerprinting attack [12, 13, 82–84] by correlating traffic patterns with known websites. Typical solutions to this problem would have Alice alter her traffic patterns while visiting the website, requiring assistance from the website or modifications to Tor itself.

Writing a Function First, Alice writes or downloads a Bento *function*: a program that is intended to be run on other Tor nodes. These functions can be powerful, but they are constrained to a limited API (§5), and run in a restricted sandbox. Critically, they run outside of unmodified Tor—in essence, they are like small servlets running on Tor relays.

Alice's function, `Browser`, is a program that takes as input a URL to download (we detail it in §7). Upon being invoked, `Browser` starts an HTTPS client, autonomously fetches the URL, saves it to a single digest file, and returns the file, padded to some multiple of bytes. Such a program can be written in about four lines of Python (see Appendix A).

Choosing Where to Run a Function Some Tor nodes opt into acting as Bento *boxes*, who are willing to run functions on behalf of other users. Like exit nodes, Bento boxes publicly specify a "middlebox node policy" of what API calls they are (not) willing to support. Alice searches the Tor directory for Bento boxes meeting `Browser`'s criteria and chooses one at random. Alice then creates a circuit to her chosen node and, with its permission, uploads and executes the function.

Composing Functions To further thwart the attacker, Alice decides to *go offline completely* during the website download by *composing* two functions together, as shown in Figure 2. She instructs the `Browser` function to also deploy, on a separate node, a simple `Dropbox` function that "puts" and "gets" a data file. `Browser` then

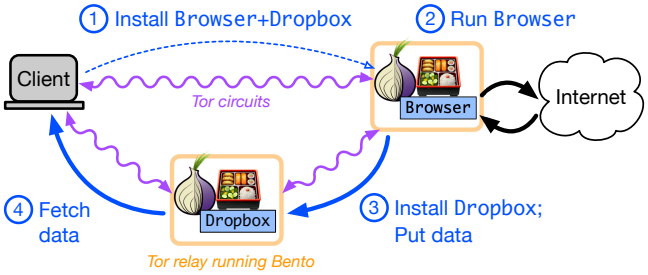

Figure 2: Example of composing two functions: `Browser` runs a web client to download a website, and `Dropbox` stores a piece of data to later be fetched.

delivers the file to `Dropbox` rather than directly to Alice, allowing her to visit the `Dropbox` node at any time to fetch the data.

From the perspective of an attacker who can sniff Alice's link, not only would she not provide activity that could be fingerprinted: she would not appear to be online at all while the website was being downloaded!

Why This Helps An attacker observing Alice's communication sees one small upload from Alice (when she installs and executes the function), followed by a large download (the padded website). Thus, because Alice is not actively involved during the download of the website, the attacker cannot gain *any* of the informative traffic dynamics that prior fingerprinting techniques require.

There are many other ways to combat website fingerprinting, but all of them require changes to Tor, clients, or web servers—an architecture like the one described here would greatly facilitate development and deployment.

Are We Nuts? This example shows that a programmable Tor would be useful, but is it worth the risk of Tor relay operators to run unvetted code (from anonymous sources)? Is it safe for users to run sensitive tasks on other users' devices? Traditional NFV has not had to address such issues, because network management is typically not performed in adversarial settings. Addressing these safety concerns (and hopefully opening up a new space to apply NFV) is Bento's central aim. We describe our design goals next.

4 BENTO GOALS

We identify five main goals that we believe are important for any programmable anonymity network:

Expressiveness We wish to empower users to write (or use) sophisticated, composable functions. To this end, we make use of a high-level programming language (Python) with no inherent limitations. To demonstrate Bento's expressiveness, we implement a wide range of functions in §9.

Protect functions from middlebox nodes We must protect users' functions against confidentiality and integrity attacks on untrusted third-party middleboxes. This is similar to the large body of work on making safe use of untrusted third-party compute resources like cloud computing [20, 34, 57, 62, 71, 79] or even Tor itself [48]. To achieve these, we employ recent advances in deploying legacy software in trusted secure enclaves [34] (see §5.4).

Protect middlebox nodes from functions We must also protect the users who run the middlebox nodes. Much like how Tor relays can express the destinations for which they wish to serve as exit nodes, middlebox nodes should be able to express policies over the actions they do and do not wish to perform on behalf of other users. Our solution is *middlebox node policies*, which allow middlebox operators to specify which system calls to permit, and how many resources to provide to functions (see §5.5). We enforce these policies by mediating access to all resources (see §5.3).

No Harm to Underlying Tor Deploying Bento should cause no degradation to the existing anonymity properties of Tor. Our functions run purely on top of Tor, and interface with it via the Stem library (see §5.3).

No Extensions to Tor We aim to sit strictly on top of Tor, and to require no additional user privileges, so as to support more robust applications. Conversely, FAN [66] permits programmability strictly within the confines of altering Tor itself. Such efforts are complementary to Bento, and could be co-deployed.

5 BENTO DESIGN

Bento is a service that clients connect to using Tor. A Bento server runs on the same machine as its companion Tor relay, but as its goal is to not modify Tor, it runs as a separate process listening on a separate port. Tor relays can provide access to their Bento server by either allowing their exit node policy to connect to the Bento server via `localhost`, or Bento may run as a hidden service. Figure 3 presents an overview of Bento's components.

5.1 Functions

At the core of Bento's programmability are user-written and provided *functions* that they can run and interact with on specific Tor relays called Bento *servers*. We have two competing goals with functions: On the one hand, they should be expressive enough to permit new, sophisticated features, services, and defenses. On the other hand, they should also be restrained from running completely arbitrary code on Tor relays or otherwise potentially compromising Tor's anonymity and security guarantees.

Bento addresses this by placing no inherent constraints on the functions' code[2]. Appendix A provides an example listing; functions can essentially be arbitrary Python in our implementation. Rather than enforce safety by limiting functions' code itself, Bento servers run functions in sandboxes, and enforce a set of policies specified by the server operators detailing what they wish to allow functions to do (e.g., some may not want to allow functions to write to disk). This combination ensures that functions can remain precisely as powerful as the Bento server operators are willing to let them be.

In the remainder of this section, we detail the aspects of the Bento server that enable safe execution and negotiation of server policies.

5.2 Bento Server

Bento servers spawn and manage a dedicated container for each client's function, and forward traffic to the appropriate container.

[2]The only constraint on functions in our implementation is that they be written in Python.

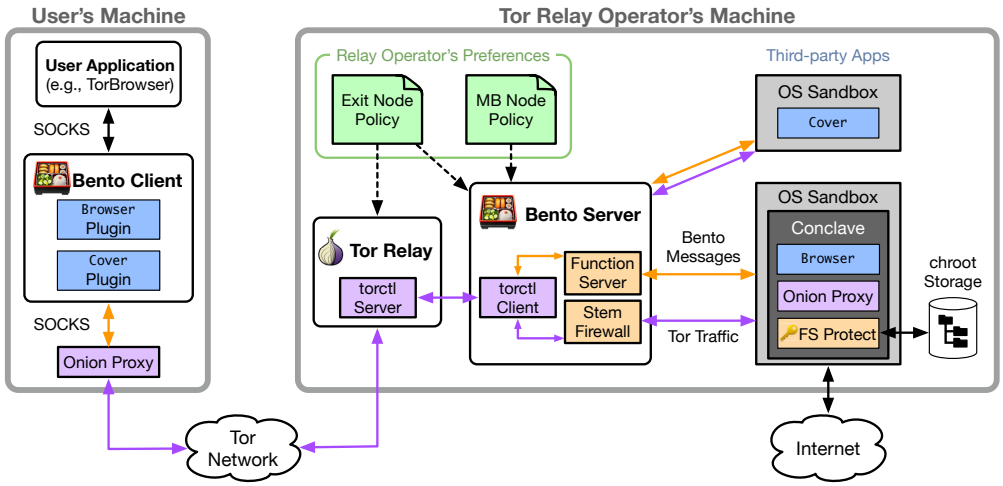

Figure 3: Design of a Bento **middlebox node.** Bento sits above an unmodified Tor, and augments it with programmable middleboxes to the Tor network. (Purple arrows denote Tor traffic; orange arrows denote Bento traffic.)

Containers protect server operators from functions that (maliciously or inadvertently) over-consume resources, access sensitive files, or interfere with other functions running on the same machine. Additionally, Bento servers expose (and mediate access to) interfaces to system resources, including an optional narrow interface to the server's Tor relay. We detail this in §5.3. Finally, Bento empowers its operators with the ability to control what sorts of tasks they will perform on behalf of others. We describe these "middlebox node policies" in §5.5.

Bento servers protect the confidentiality and integrity of a client's function and associated data from the untrusted Bento operator by allowing them to execute in Intel SGX enclaves [6, 35, 54] within the container. This has the added benefit of providing plausible deniability to the Bento operators with respect to a function's processing of abusive content. We describe this in §5.4.

5.3 Container Management

Bento operators are responsible for providing container images. This design choice allows operators basic control over the types of functions that may run, and avoids the overhead of clients uploading containers. The images themselves are not applications *per se*, but rather servers that communicate with the Bento client to handle the post-launch provisioning, loading, and execution of the client-provided function.

Initializing and Shutdown When a Bento client connects to a Bento server, it requests a container image. The server spawns the container and returns to the client two tokens: an *invocation token* and a *shutdown token*. From this point onward, the client includes the invocation token in subsequent messages, and the server forwards the messages to the corresponding server running in the container.[3]

The Bento server terminates and reclaims the container's resources either when the container's function terminates, or the client presents the shutdown token. The distinction between invocation and shutdown tokens allows a client to share the invocation token (and thus, use of the function) with other users while retaining exclusive shutdown rights. Regardless, Bento functions fate-share with the middlebox nodes they run on, and thus must assume that the function may ungracefully terminate at any time.

Sandboxing and Resource Accounting Bento servers use Linux's cgroup and namespace features to provide containers with a nominal amount of memory and limited space in a `chrooted` file system, so that clients cannot access any files but their own. An operator may further manage these resource limits in aggregate over all container instances, ensuring that the co-resident Tor relay maintains a set minimum portion of the machine's total resources.

To ensure that functions cannot violate a Tor relay's exit node policies, the Bento server converts the exit node policies into analogous iptable rules, and applies these rules to each container. Note that, if the relay does not wish to run as an exit node, then this would similarly preclude all functions it runs from direct network access, and functions would be strictly limited to communicating via Tor circuits. Bento also permits operators to apply system call filters in the form of seccomp policies [26] to disallow a function's use of specific system calls, such as `fork` and `execve`.

Container Interface to Tor Instance We envision policies that middlebox node operators may wish to enforce that cannot be satisfied using OS-level sandboxing techniques alone. For instance, functions may use the popular Stem library [75] to programmatically create circuits and launch hidden services. To permit safe, shared access to Stem, Bento includes as part of its policy enforcement layer a Stem "firewall" to which functions must connect (via a local socket) to issue all Stem invocations. The firewall maintains state about the circuits each function is allowed to access, and the Stem routines the function may invoke.

[3]We note that tokens can be blinded, especially with the use of an enclave, but we leave that for future work.

5.4 Standard Container Images

Although operators may advertise any image for running a client's function, we envision two standard images that collectively handle a broad set of use cases.

Python container The *Python container* provides an execution environment for a client-provided Python application. This container targets cases where the client's function does not process sensitive information, and is Bento's most direct analog to FAN's proposed use cases [66]. For instance, the Bento client and function might further encapsulate the source-to-destination TLS stream with a padding scheme to foil deanonymization attacks. This container also allows non-sensitive network measurements, such as of the latency or bandwidth to a Tor relay or destination server.

Python-OP-SGX container The *Python-OP-SGX container* is similar to the Python container, but instead executes the Python application, as well as an optional companion Onion Proxy, in Intel SGX enclaves by using a library OS for running unmodified applications in SGX [34, 80]. In this way, the container guarantees the confidentiality and integrity of the application's memory.

This container targets two broads use cases: (1) the function is a specialized application-level proxy, such as a web proxy that caches content for future client download or that transforms the plaintext content so as to avoid fingerprinting attacks, or (2) the function itself is a hidden service. Since a function, acting as a hidden service, has keying material that it must share with a Tor instance, the container cannot safely use the operator's Tor instance to manage the hidden service, and thus the container allows the function to spawn a dedicated Onion Proxy.

Our implementation specifically makes use of *conclaves* ("containers of enclaves") [34], which allow for arbitrary legacy applications to be deployed within a set of interconnected enclaves, where some enclaves securely provide traditional OS services on behalf of the enclaved application, such as an encrypted and integrity-protected filesystem. In Figure 3, we refer to such an enclaved filesystem as *FS Protect*.

FS Protect generates an ephemeral encryption key when the filesystem is launched in an enclave; the container ensures that the enclaved filesystem is the only writable filesystem available to the function, and therefore that all filesystem writes are encrypted. Prior to function execution, the Bento client attests the container's image and establishes a secure TLS channel to the container's function loader; the Bento client then uploads the function, and any associated data to copy to FS Protect, over this channel. In addition to securing the client's on disk content, FS Protect simultaneously provides the Bento operator with plausible deniability in the event that the function uses abusive content, as, much like Tor traffic, the function's execution and on-disk resources are unobservable by the operator.

Attestation Before uploading its function, a Bento client gains assurance that the Bento server is running correctly by using SGX's remote attestation feature [42] to verify that the application is truly running inside an enclave as well as check the current TCB version of the remote system to see if it has been patched against known vulnerabilities. To assure a client that it is operating correctly, a Bento box generates an attestation verification report when it first loads the Bento server by creating a quote and sending it to the Intel Attestation Service (IAS).[4] Note that the only code needing attestation is the Bento execution environment (including Python), not the individual user functions.

We envision two paths for client verification of this attestation. Traditionally, the server generates an attestation report and returns the report to the client, who could then present the report to IAS for verification. This can be done at any time before a client loads the function, preventing any correlation between client and function load. Alternatively, the Bento server can perform this verification and, similar to OCSP stapling [1], return to the client both the report itself as well as Intel's verification of the report. Thus, Intel (and others) only learns which relays are running conclaves (which is already in the public directory).

5.5 Middlebox Node Policies

Allowing other users to run custom software on one's own machine has obvious risks, particularly in the context of an anonymous network such as Tor. For any such architecture to be viable, it must give users power to assert what they are and are not willing to do on behalf of others. Tor itself runs into this challenge with respect to exit nodes: not all users are willing to connect to any service on any machine. Tor's solution to this is exit node policies: fine-grained policies of which IP addresses and ports it will or will not visit on behalf of other Tor users.

We borrow this idea and introduce *middlebox node policies*. At a high level, these are similar to exit node policies: middleboxes specify what they are and are not willing to perform on behalf of others. The primary difference is the set of actions the policies themselves span.

Middlebox Node Policy Design Bento's middlebox node policies are boolean values over the set of API calls that Bento exposes to functions. Every system call and Stem library function that can be exposed to functions is also specified in the middlebox node policy. This is similar in spirit to how Android applications obtain permissions: users can specify (to some extent) the resources the applications may have access to, but beyond that are not able to assert policy over the internal workings of the applications.

Our architecture's design is not strictly bound to this specific choice of policy space. There are alternative designs, such as requiring that certain API calls be invoked only from functions within an enclave, or incorporating taint tracking and restricting network calls based on the flow of tainted data, and so on. Ultimately, we believe that a more comprehensive discussion among the Tor developer and user communities will be necessary in finalizing the policy space.

Disseminating Middlebox Node Policies We envision that middlebox node policies could be disseminated as part of the Tor directory, as with exit node policies. However, such integration with Tor is not strictly necessary. To support immediate, incremental deployment, we have implemented a function that runs on a well-known port that returns the node's middlebox node policy, allowing users to query Bento nodes to see what they support.

Function Manifest Files When a user sends a function to a Bento server, the user includes the function's *manifest file*, similar

[4] We will not go into the full details of the attestation process here but encourage the interested reader to see [37].

in spirit to an Android app manifest. Upon receiving the manifest, Bento compares it to its own middlebox node policy; if the manifest asks for more permissions than the node's policy permits, then the function is rejected. Otherwise, the Bento server sets up the execution environment, and constrains the sandbox or conclave to permit only the specific API calls that the manifest file requested (even if the middlebox policy allowed for more).

6 BENTO'S SECURITY PROPERTIES

We evaluate how Bento achieves its security goals from §4.

6.1 Attacks Against Functions

One of our primary goals is to protect functions from middleboxes, and from other functions running on those middleboxes. We envision two classes of attack:

The first class involves altering or exfiltrating data or code as it executes. Conclaves give Bento strong guarantees of confidentiality and integrity [34], subject to the reliability of the trusted hardware. Data stored (and code run) in a conclave is protected against inspection and tampering from both other applications and an adversary with physical access. Thus, even if a middlebox and adversarial function were to collude, the code and data are protected.

Alternatively, an attacker might try to inject packets into a function that he himself does not control. We prevent this attack through the invocation token provided on function load. This token is then required to direct any further communication or requests to the running function (see §5.3).

6.2 Attacks Against Middleboxes

We envision several concerns that a Bento middlebox operator might have about its safety. We discuss them here, along with how our design helps to prevent them.

Running arbitrary code One of the benefits of Bento is its ability to allow for arbitrary code. This of course brings with it concern that third-party programs may run amok. Bento does not seek to limit what a third-party program can do *within* a container, but rather what side-effects it can have on the system itself. As described in §5.5, Bento achieves this with middlebox node policies and function manifests.

Resource exhaustion attacks A malicious function could try to consume a large amount of resources on a Bento box. Our OS-level sandboxing mitigates this, as it allows us to restrict the maximum level of resource consumption of the processes running in the sandbox.

To work around the resource restrictions on a *single* function, an adversary might try a denial of service attack on the middlebox by rapidly flooding the middlebox with a large number of functions. We prevent this from starving out the rest of the middlebox and its processes through the OS sandbox as well, by limiting the *total* resource consumption of Bento to a specified amount.

While Bento is able to limit the total resources that any given function can consume, it does not yet have a mechanism for ensuring fairness amongst users. This brings several concerns, such as a malicious user preventing others from loading functions, leveraging functions (and the middleboxes' resources) as a tool for undertaking DDoS attacks, and using functions to modulate the middlebox's CPU or network bandwidth so as to affect the traffic of others. However, there are many existing lines of work in this space that we believe are promising and would also apply here, such as proofs of work [9, 25], anonymous credentials [18], or combinations thereof [14, 21, 31]. We leave these to future work.

Abusive functions Bento has several safeguards to protect against abusive functions. For instance, by adopting the relay's exit node policy (§5.3), Bento restricts the parties with which a function can communicate.

The primary remaining potential for abuse is that of storing illicit content on the operator's machine. In the most extreme case, an operator can protect themselves by setting a policy that prevents functions from accessing the filesystem or storing *any* data on the node. However, this greatly restricts the functions that could be run on such a node. Alternatively, the operator can allow functions to execute in the Python-OP-SGX container (described in §5.4), which encrypts all filesystem writes with an ephemeral key inaccessible to both the operator and the function itself. As a result of this design, a Bento operator can only ever access encrypted data, resulting in the same level of protection and plausible deniability as Tor currently provides to standard relay operators. Thus we believe that Bento does not make Tor relay operators more susceptible to abuse.

6.3 Attacks Against Users

As many users employ Tor specifically to protect their anonymity, we must be careful that Bento does not compromise this. To discuss the various deanonymization challenges a user might face (and why Bento protects against them), we will briefly walk through the life cycle of a user's interaction with Bento.

A user must first fetch the middlebox node policy for its chosen Bento server. This could be done over a Tor circuit if the user wishes to hide that she is using Bento.[5] She also obtains the attestation verification report, to ensure that the node is setup correctly. We discuss how we would maintain a user's privacy when validating this attestation in §5.4.

Next, the user uploads her function. Note that a node operator should not be able to link function uploads to a specific user (or even identify them) as there is nothing fingerprintable about uploads in terms of the code itself. For maximum privacy, function uploads could also be encrypted and only decrypted within the enclave. The node then returns the necessary tokens to the user, which could be blinded for privacy as we discuss in §5.3.

When the user wishes to run her function, she sends her messages to it with the (blinded) invocation token. We note that preventing a malicious operator from fingerprinting a function's system call patterns, network calls, and the like in order to identify what function a user is running (or identify that two separately uploaded functions are actually the same) is outside of the scope of this paper. We believe techniques that build oblivious filesystems for SGX (such as [2]) would be applicable here. While a network adversary might be able to tell that a Bento node is running a function, he

[5]We will subsequently assume that all user interactions with the Bento node are done over Tor circuits if the user wishes to preserve strong anonymity.

will not be able to link this function back to the specific user who uploaded and invoked it, as a privacy-conscious user only interacts with the node over Tor circuits.

Finally, when the user is done with her function, she uses the (blinded) shutdown token to terminate it.

One final attack on users we consider is that of an "adversarial" function, such as one that seeks to deanonymize a user or identify a specific Tor node they are using by running a function that tries to affect their traffic in some way, rendering it easier to fingerprint. This could be achieved, for example, by creating a function that over-consumes CPU or bandwidth on the node, thus slowing down any traffic which passes through only that specific node, or trying to "tag" another user's traffic in some way. While it is difficult to rule out all possible variants of this attack, there are two key properties of Bento that greatly mitigate these threats. First, Bento provides strong sandboxing and isolation when executing processes, preventing a function from introspecting on or affecting another user's data or network traffic. Second, Bento servers can restrict how many resources a function is permitted to use (see §5.3 and §6.2), preventing starvation and other side-effects. Together, these safeguards help prevent an untrusted function from affecting the resource consumption of another or from trying to starve a co-resident process.

7 CASE STUDY: BROWSER

In a *website fingerprinting* attack, an adversary is able to view all traffic to and from a victim client. The attacker uses the patterns of packet transmissions as a fingerprint of the website. For websites with mostly static content, these patterns can be an effective fingerprint, allowing the adversary to uniquely identify which website the user is visiting.

Although Tor users can take some measures to prevent website fingerprinting attacks, no existing solution effectively prevents an adversary from deriving salient features from users' web browsing traffic. Recently, Tor has integrated preliminary mechanisms into the protocol to introduce dummy traffic and confuse most fingerprinting techniques through traffic padding. However, these solutions increase the load into and out of the Tor network, while not introducing sufficient noise into the contents of web traffic.

7.1 Prior approaches

Past efforts in securing Tor web browsing against fingerprinting attacks largely propose making significant changes to the underlying Tor source code. WTF-PAD [45] introduces adaptive padding techniques to optimize bandwidth by padding Tor traffic in low usage streams. Additionally, Walkie-Talkie [84] makes use of half-duplex communication such that the server must reply to the client in non-overlapping bursts, thus shielding identifiable web traffic behaviors.

7.2 Function overview

The insight behind the `Browser` function is that the adversary cannot observe identifiable behaviors if the user is not the one running the web client! `Browser` runs the web client on a separate Bento box (an exit node, in this case). The function then packages up the entire webpage and ships it back to the client. The size of the page alone can reveal information about it, so `Browser` pads this up to a given multiple of bytes. Both the URL to fetch and the size to pad to are provided by the client when invoking the function. We present the code to `Browser` in Appendix A.

This is immediately deployable without changes to Tor or to the Tor Browser. As shown in Figure 3, our implementation creates a basic application-layer web proxy at the exit node of the user's circuit. Although the current design of of `Browser` is not suitable for latency-sensitive interactions (e.g., video chatting or online gaming), it can easily be adapted to support cookies for interactive browsing.

Attack Accuracy	Defense
93.9%	None (unmodified Tor)
69.6%	`Browser`, 0MB padding
8.25%	`Browser`, 1MB padding
0.0%	`Browser`, 7MB padding

Table 1: Accuracy of Deep Fingerprinting [73] attacks against unmodified Tor and `Browser` with varying amounts of padding. `Browser` offers significant defense.

7.3 Evaluation

We evaluate three aspects of the `Browser` function: its efficacy as a website fingerprinting defense, its performance in terms of page load time, and its scalability.

`Browser` as a website fingerprinting defense We first evaluate `Browser` under the adversarial conditions showcased in prior work [73]. We used the same experiment setup as Sirinam et al. [73]: 10 medium-sized Amazon E2 instances, each running a Bento client, and all Tor traffic between the client and its guard relay is recorded. We visited 100 popular websites [4] at least 10 times using a standard Tor browser and again using `Browser` (with 0MB, 1MB, and 7MB padding of random bytes), running inside of an SGX-based conclave. We apply a sophisticated fingerprinting attack involving deep learning [73]. To train the deep learning models, we used 5 NVIDIA 1070 and 2 NVIDIA 1070ti GPUs.

We summarize our results in Table 1. Unmodified Tor results in a 93.9% accuracy for an attacker. Merely by using `Browser`—without any padding—the change in traffic patterns results in a significant decrease in attacker accuracy, down to 69.6%. Adding even a nominal amount of padding (1MB) results in a drastic decrease to 8.25% accuracy, and a large amount of padding (7MB) renders the attack completely ineffective (0% accuracy).

The reason `Browser` is so effective is because it fundamentally removes the benefits of being an adversary close to the client; to the attacker, the traffic patterns appear to be a modest upload (installing the function), and then a pause (while the webpage is loaded at the Bento box), and a long stream of packets back. This removes much of the information that fingerprinting attacks rely on. We believe this points to a broader class of promising functions: those that offload tasks from a client into *autonomous agents* who can act on the user's behalf from other nodes within the network.

Performance of `Browser` Our performance benchmarks for `Browser` focus on the time to fully download a webpage—from

Domain	Standard Tor	Browser 0MB	Browser 1MB	Browser 7MB
indiatoday.in	5.0	6.4	34.9	86.0
yahoo.com	6.7	**6.3**	21.2	87.4
netflix.com	8.5	**8.1**	28.4	86.3
ebay.com	6.1	7.0	22.3	81.8
aliexpress.com	3.1	5.9	37.7	91.9

Table 2: Download times (in seconds). Bold numbers denote instances where `Browser` performs faster than standard Tor. Note that users can obtain the viewable webpage in the time to download the 0MB version; the additional download time is purely for padding.

the time the client issues the request to the function until it is done downloading, with various padding levels.

Table 2 shows our results. At a high level, it is mostly what one might expect: larger websites or large amounts of padding increase the download time. With greater padding comes greater security, but also a decrease in performance—a direct result of the "anonymity trilemma". However, for smaller websites, the time to download the entire website using `Browser` can sometimes actually be *lower* than traditional Tor. This is because TCP's performance for small file transfers is heavily affected by round-trip time (RTT); in traditional Tor, this is the circuit's RTT plus the exit node-to-server RTT. For Bento, it is just the latter.

Note further that the results represent the *full* time to download the content, but because the padding comes *after* the webpage, the client receives (and can render) the webpage in the amount of time it takes to transfer under the 0MB case. It is important, however, that the user's subsequent actions be delayed until after the full padding has downloaded.

Scalability of `Browser` We provide a brief analysis of the scalability of `Browser` (and other functions), as well as of Bento as a whole.

One of the main potentials for overhead in our architecture is the use of SGX and conclaves. A thorough microbenchmark analysis of conclaves and SGX overhead was performed in [34], so we do not repeat them here. However, we summarize some of the relevant results for completeness. The authors of [34] note that the time to swap in and out of the conclave introduces nominal overheads for running a "CDN" (a highly latency-sensitive application); these overheads would be even less impactful for Tor, as it already experiences high latency overheads from the Tor circuits themselves. We also note that the page load times from the previous section were measured while using SGX, and, in some cases, the download time is actually lower. Thus, the use of conclaves does not provide a significant performance impact.

A second notion of scalability is the number of functions that can run on a single node. SGX provides a limited amount of protected memory (128MB), with only 93MB of this usable by applications [34], meaning that we are constrained in the number of functions that can be running concurrently on a node. To quantify this, we

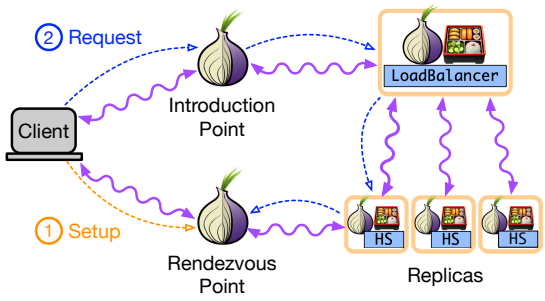

Figure 4: The `LoadBalancer` function forwards requests from the Introduction Point to one of its hidden service replicas. In our implementation, `LoadBalancer` automatically scales the replicas up and down to meet demands.

estimate[6] the memory required by Bento and `Browser` by running them outside of conclaves. The maximum memory usage of a Bento server and `Browser` is roughly 16–20 MB, depending on the webpage being downloaded. We add to that the estimated 7.3 MB required for conclaves to obtain the total memory footprint of running the `Browser` function in Bento.

We make a few observations about these results. First, Bento is implemented in Python with little concern for memory footprint, so it is likely that these numbers would be easily improved if memory usage is a concern. Second, even without optimization, we are already able to run multiple functions without straining the SGX memory limits (we additionally note that the memory required by a function is highly dependent on which function one is running). Third, SGX has support for paging; as we do not expect all functions loaded on a node to always be running, enclaves could be paged out if they are not currently being invoked. In summary, we do not believe memory usage is a fundamental limitation of the architecture or a barrier to scalability.

8 CASE STUDY: HIDDEN SERVICE LOAD BALANCER

In the "non-anonymous" Internet, it is common to replicate popular servers, dynamically scale them up and down as demand varies, and use a load balancer middlebox to direct requests to the least loaded servers. This is surprisingly difficult in Tor. Here, we show how to easily do it with Bento.

8.1 Prior approaches

Load balancing over anonymity networks has been introduced previously. PeerFlow [40] proposes load balancing for Tor relays through a bandwidth-weighted voting process, limiting the ability of an adversary to fake the bandwidth.

For hidden services, OnionBalance [59] introduces the notion of pre-creating replicas and publishing a different descriptor (and set of Introduction Points) to the hidden service directory for each replica. The default Tor client will randomly choose an Introduction Point from those descriptors, thus evenly distributing clients across replicas. This unfortunately makes it heavily dependent on

[6]It is difficult to provide exact measurements for memory usage within SGX because of its inherent security properties and design.

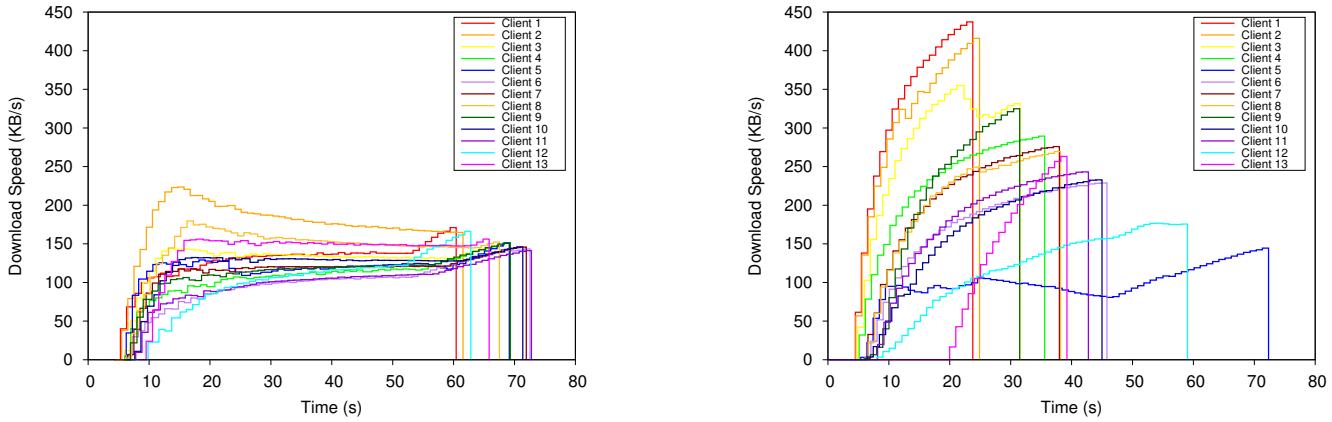

Figure 5: Per client bandwidth with and without our `LoadBalancer` function. The plot on the left shows per client download speed without the `LoadBalancer` while the plot on the right demonstrates the benefit from utilizing our function. As more clients access the hidden service, additional replicas are spun up to handle the load.

the maximum number of introduction points available, as well as limiting adaptability and exhibiting slow failover [76].

To improve the scalability of hidden services, Sucu introduces additional techniques for load balancing [76]. The primary technique performs balancing at the circuit level and requires modifications to the Tor source code.

8.2 Function overview

`LoadBalancer` can spin up new hidden service replicas and direct client requests to (or away from) replicas to distribute load and improve performance. Similar to how hidden services operate today, `LoadBalancer` establishes introduction points and listens for clients' incoming requests to join them at a rendezvous point. However, rather than connect to the rendezvous point itself, `LoadBalancer` chooses from a set of replicas (or spins up a new replica) and instructs the replica to connect to the rendezvous point on its behalf. To create a replica, the `LoadBalancer` copies all files (including the hostname and private key) to the new instance; this motivates deploying `LoadBalancer` within conclaves. Figure 4 provides an overview.

`LoadBalancer` receives periodic messages from replicas describing their load, and uses high- and low-watermark thresholds to determine when to create or remove a replica. Replica creation is transparent to clients: there is but one set of introduction points (that the `LoadBalancer` establishes), and, naturally, clients never learn the identities of the hidden service nodes. Our current selection process randomly chooses a replica from all active middlebox nodes, but this could be more sophisticated, taking into consideration properties like geography, latency, etc.

8.3 Evaluation

We evaluated `LoadBalancer` with multiple replicas of our hidden service running on the Tor network and varying load on the hidden service.

Our experiments are performed with four Tor nodes that host the hidden service (Amazon EC2 T2 instances with 2 vCPUs, 4 GB RAM and Ubuntu 18.04 OS) and thirteen clients (Amazon EC2 micro instances with Ubuntu 16.04, 1 GiB Memory, 1 CPU). Clients arrive at roughly 1sec intervals, and each client downloads a 10 MB file from the hidden service. We perform the experiment with and without `LoadBalancer`, keeping the environment consistent across runs.

Figure 5 shows the results. Without the `LoadBalancer`, we can see that the download speed of each client reaches roughly the same maximum, as the clients share the single server's bandwidth, and the clients take roughly the same amount of time to download the file. With `LoadBalancer` configured to permit at most two clients at a time, we observe considerably improved use of resources. At its peak, a total of four machines (the original plus three replicas) service the clients' requests, resulting in shorter download times, more dedicated use of each replica's bandwidth, and better performance overall.

9 OTHER FUNCTIONS

Here, we demonstrate Bento's breadth in solving a wide range of problems by briefly describing other functions we have implemented. These span a wide range of application domains, and some of them address longstanding questions within the Tor community.

9.1 Cover Traffic

Anonymity systems that offer "strong anonymity" [22] send *cover traffic* whenever there are hosts with nothing to send. This ensures that the size of the anonymity set (the set of principals who could have taken a given observed action) is as close to the entire set of participants as possible. Tor explicitly chose not to do this, instead preferring efficient, low-latency use of its collective resources, under the assumption that if it was fast and easy enough to use, then increased usage would naturally create additional cover traffic.

Unfortunately, that is not always the case. Circuit fingerprinting and website fingerprinting attacks are facilitated by the fact that there is not always sufficient cover traffic.

Function overview `Cover` instructs a Bento box to ensure that a given circuit always transmits at a fixed rate, sending junk traffic if it has no legitimate traffic to send.

To see the strength of this simple primitive, consider how it could compose with `Browser`: a client could establish bidirectional cover traffic with an exit node, then initiate the download of a webpage. Instead of waiting until the webpage is fully downloaded to begin sending it, the Bento box could send back immediately, permitting support for interactive webpages without sacrificing anonymity.

9.2 Dropbox

In developing other functions, we found it useful to be able to ephemerally store files in the Tor network. Allowing users to store files without having to remain online gives them the flexibility to thwart attackers by going offline without having to interrupt their computation. More generally, it provides a level of indirection that is useful in composing functions, making this deceptively simple function quite powerful.

Function overview After a user installs a `Dropbox` function on a middlebox, the function operates in two phases. The first phase accepts a *put* request, along with the invocation token, which serves as a capability [88] permitting access to that dropbox. When `Dropbox` receives a put request with the appropriate token, it saves the data in the function's `chroot`ed directory and, if written to disk, using an ephemeral key stored in the enclave (§5.4). The second phase permits *get* requests with the same invocation token, up to either some maximum amount of bandwidth, number of requests, or expiry time, after which the function deletes the file and terminates.

9.3 Shard

Functions like `Dropbox` provide a level of indirection: users can store a file at one point in time, keep it stored in the network, and return at a later time to retrieve it. Although powerful, it also introduces some potential concerns: Suppose an adversary's machine were used to run the `Browser` function in Figure 2, outside of a conclave. Then, the adversary would not know the client who initiated the connection, but he would know the location of the `Dropbox` function. If the stored file is only available at that one *known* location in the network, then it puts the attacker in a potentially powerful position to launch a fingerprinting attack. Additionally, with only a single copy of a file, if the machine hosting `Dropbox` crashes, then the file will be lost.

Function overview The `Shard` function addresses these concerns by applying a digital fountain approach [11, 53] to spread a single file across multiple machines. It takes as input a file, a number of shards N to create, and a minimum number necessary to reconstruct the file, $1 \leq k \leq N$. `Shard` uses standard linear encoding techniques to ensure that retrieving any k of the N shards suffices to reconstruct the file. (In the trivial case where $k = 1$ and $N > 1$, `Shard` simply replicates.) `Shard` then deploys these shards by invoking the `Dropbox` function on other machines. When the user is ready to obtain the files, it chooses a subset of drop-off locations.

This empowers users in three ways. First, if any of the `Dropbox` nodes fail, then the file is still available. Second, if a user subsequently learns that certain regions of the network are less trustworthy or more susceptible to attack, then she has flexibility over where she accesses the data. Finally, using multiple shards increases the probability that one user's shards are at the same location as other users', resulting in additional cover traffic.

9.4 Future ideas

Multipath routing An open research question in Tor is how to efficiently use the network's overall bandwidth, as well as adapt to traffic congestion. Several works [5, 87] propose adding a multipath routing scheme that splits a stream across multiple circuits sharing a common exit relay, and that dynamically schedules traffic over the stream's circuits based on their throughput. Rather than modify the Tor code base, we are exploring whether multipath routing designs can be implemented as Bento functions.

Geographical avoidance Prior work has introduced *provable avoidance routing* [49, 51, 52]: allowing users to specify geographic regions where packets should not traverse, and then providing proof that the packets did not go through such regions. However, as these techniques rely on *end-to-end* RTT measurements and knowledge of the location of each Tor relay on the circuit, they are not immediately applicable to hidden services, wherein no one entity (neither the source nor hidden service) knows the *entire* end-to-end circuit. We are exploring whether functions, running inside an enclave at the rendezvous point, enable computing the proofs of avoidance while maintaining privacy.

Hidden service DDoS defense To mitigate DDoS attacks against hidden services, Tor supports client authentication to the HSDirs and introduction points—solutions that are only appropriate for *private* hidden services. A number of proposals [23, 60, 63, 64] recommend additional defenses that change the topology of the introduction points, add new cell types to assist in rate limiting, or require client-side proofs of work prior to establishing a connection. We are exploring whether these approaches can be implemented as function-specific protocols, rather than modifying Tor's existing protocols.

10 ETHICAL CONSIDERATIONS

As our experiments were performed over the actual Tor network, we were careful not to impact the security and privacy of its users or the performance of the network. We ran our own exit relays and hidden service servers (as this is where our Bento functions are deployed), and experimented with fingerprinting only our own traffic.

11 LIMITATIONS AND FUTURE WORK

This paper might raise more questions than it answers; we believe that is a good thing! In seeking to achieve properties that would be much more complicated or inefficient in the current design of anonymity networks like Tor, we allowed ourselves to take rather drastic departures. As a result, we have identified important goals for any programmable anonymity network, and have provided a proof-of-concept architecture that achieves them.

That said, we acknowledge there is a considerable amount of future work necessary for programmable anonymity networks to become a practical reality. Though we have discussed many of Bento's limitations throughout the paper, we summarize them here, along with a few others that were not previously discussed. We believe that many of the existing limitations provide interesting avenues for future work.

Reliance on a TEE One limitation is that Bento explicitly includes a trusted execution environment in its trusted compute base (Intel SGX in our implementation). If the TEE were compromised, what would that mean for Bento? In the worst case, an attack on the TEE would allow a Bento operator to possibly introspect on running functions, view a user's data as the function executes, or alter the execution of a function—a complete breakdown of anonymity for some functions. Moreover, the Bento operator might also lose access to the plausible deniability that we discuss in the case of abusive content.

These concerns are mitigated somewhat by the fact that Bento permits flexibility in which TEEs users are willing to trust. Bento is built on top of conclaves—which are not strictly bound to SGX—and thus we too can work with any TEE that has similar properties.

Even without access to a TEE, we believe the programmable anonymity networks and Bento can still be useful. Many basic functions, such as `Cover`, that do not contain sensitive information could potentially be executed even without a TEE (see our discussion in Section 5.4). We also note that trusted hardware is just one way to achieve the properties of a TEE. One interesting avenue for future work would be to explore if there are other ways to achieve our stated goals without using a TEE, such as with computation over encrypted traffic [62, 71].

Lack of fairness Another limitation that we have previously discussed is that Bento currently does not have a mechanism implemented for ensuring fairness among users. This leads to several concerns in the case of malicious users, such as loading numerous functions on many Bento servers to prevent others from using the system or leveraging fuctions and middlebox resources to carry out DDoS attacks. We believe that this is an interesting area of future work, and that many existing research areas such as proofs of work [9, 25], anonymous credentials [18], or combinations thereof [14, 21, 31] are potentially promising.

Unclear incentives for adoption In this paper we do not solve a number of questions pertaining to incentives for users to run Bento, nor for the Tor community at large to adopt it. Perhaps the best way to answer this question is a further demonstration of the power of programmable anonymity networks: What else can be done with Bento? What other problems can be solved, features can be introduced, or spaces in the anonymity trilemma explored once one has access to a programmable Tor network? This paper is the first step towards programmable anonymity networks; to facilitate further work in this vein, we have made our code publicly available.

12 CONCLUSION

We introduced *programmable anonymity networks* and showed them to be possible and useful. We showed with a series of applications that even simple programs running on nodes in the Tor network can result in significant improvements to security, performance, and resilience. We also addressed the elephant in the room—the natural concerns of running untrusted code on other users' machines—and presented an architecture that leverages recent results in trusted execution environments to ensure safety for both users and relay operators. We view this paper as the first step towards programmable anonymity networks, and hope that it gives rise to exploration of new domains in which to apply NFV. To assist in these future efforts, we have made the Bento code publicly available at `https://bento.cs.umd.edu`.

ACKNOWLEDGMENTS

We thank our shepherd, Barath Raghavan, and the anonymous reviewers for their helpful feedback. This work was supported in part by NSF grants CNS-1816422, CNS-1816802, and CNS-1901325.

REFERENCES

[1] D. Eastlake 3rd and Huawei. 2011. Transport Layer Security (TLS) Extensions: Extension Definitions. RFC 6066. (Jan. 2011). `http://www.ietf.org/rfc/rfc6066.txt`
[2] Adil Ahmad, Kyungtae Kim, Muhammad Ihsanulhaq Sarfaraz, and Byoungyoung Lee. 2018. OBLIVIATE: A Data Oblivious Filesystem for Intel SGX.. In *Network and Distributed System Security Symposium (NDSS)*.
[3] Masoud Akhoondi, Curtis Yu, and Harsha V. Madhyastha. 2013. LASTor: A Low-Latency AS-Aware Tor Client. In *IEEE Symposium on Security and Privacy*.
[4] Alexa Top 500 Global Sites [n. d.]. Alexa Top 500 Global Sites. `http://www.alexa.com/topsites`. ([n. d.]).
[5] Mashael AlSabah, Kevin Bauer, Tariq Elahi, and Ian Goldberg. 2013. The Path Less Travelled: Overcoming Tor's Bottlenecks with Traffic Splitting. In *Privacy Enhancing Technologies Symposium (PETS)*.
[6] Ittai Anati, Shay Gueron, Simon Johnson, and Vincent Scarlata. 2013. Innovative technology for CPU based attestation and sealing. In *International Workshop on Hardware and Architectural Support for Security and Privacy (HASP)*.
[7] James Anderson, Ryan Braud, Rishi Kapoor, George Porter, and Amin Vahdat. 2012. xOMB: Extensible Open Middleboxes with Commodity Servers . In *ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*.
[8] Bilal Anwer, Theophilus Benson, Nick Feamster, and Dave Levin. 2015. Programming Slick Network Functions. In *Symposium on SDR Research (SOSR)*.
[9] Adam Back. [n. d.]. Hashcash. `http://www.cypherspace.org/hashcash/`. ([n. d.]).
[10] Alex Biryukov, Ivan Pustogarov, and Ralf-Philipp Weinmann. 2012. TorScan: Tracing long-lived connections and differential scanning attacks. In *Computer Security–ESORICS 2012*. Springer, 469–486.
[11] John W. Byers, Michael Luby, Michael Mitzenmacher, and Ashutosh Rege. 1998. A Digital Fountain Approach to Reliable Distribution of Bulk Data. In *ACM SIGCOMM*.
[12] Xiang Cai, Rishab Nithyanand, Tao Wang, Rob Johnson, and Ian Goldberg. 2014. A Systematic Approach to Developing and Evaluating Website Fingerprinting Defenses. In *ACM Conference on Computer and Communications Security (CCS)*.
[13] Xiang Cai, Xin Cheng Zhang, Brijesh Joshi, and Rob Johnson. 2012. Touching from a Distance: Website Fingerprinting Attacks and Defenses. In *ACM Conference on Computer and Communications Security (CCS)*.
[14] Jan Camenisch, Susan Hohenberger, Markulf Kohlweiss, Anna Lysyanskaya, and Mira Meyerovich. 2006. How to win the clonewars: efficient periodic n-times anonymous authentication. In *ACM Conference on Computer and Communications Security (CCS)*.
[15] Frank Cangialosi, Dave Levin, and Neil Spring. 2015. Ting: Measuring and Exploiting Latencies between All Tor Nodes. In *ACM Internet Measurement Conference (IMC)*.
[16] B. Carpenter. 2002. *Middleboxes: Taxonomy and Issues.* RFC 3234.
[17] Sambuddho Chakravarty, Angelos Stavrou, and Angelos D Keromytis. 2010. Traffic analysis against low-latency anonymity networks using available bandwidth estimation. In *Computer Security–ESORICS 2010*. Springer, 249–267.
[18] David Chaum. 1985. Security without identification: transaction systems to make big brother obsolete. *Commun. ACM* (1985).
[19] David Chaum. 1988. The Dining Cryptographers Problem: Unconditional Sender and Recipient Untraceability. *Journal of Cryptography* 1, 1 (1988), 65–75.
[20] Michael Coughlin, Eric Keller, and Eric Wustrow. 2017. Trusted Click: Overcoming Security issues of NFV in the Cloud. In *ACM International Workshop on*

Security in Software Defined Networks & Network Function Virtualization (SDN-NFVSec).

[21] Ivan Damgård, Kasper Dupont, and Michael Østergaard Pedersen. 2006. Unclonable group identification. In *International Conference on Theory and Applications of Cryptographic Techniques (EUROCRYPT).*

[22] Debajyoti Das, Sebastian Meiser, Esfandiar Mohammadi, and Aniket Kate. 2018. Anonymity Trilemma: Strong Anonymity, Low Bandwidth Overhead, Low Latency—Choose Two. In *IEEE Symposium on Security and Privacy.*

[23] Alex Davidson, Ian Goldberg, Nick Sullivan, George Tankersley, and Filippo Valsorda. [n. d.]. Privacy Pass. ([n. d.]). `https://privacypass.github.io/`.

[24] Roger Dingledine, Nick Mathewson, and Paul Syverson. 2004. Tor: The Second-Generation Onion Router. In *USENIX Security Symposium.*

[25] Cynthia Dwork and Moni Naor. 1992. Pricing via processing or combatting junk mail. In *International Cryptology Conference (CRYPTO).*

[26] Jake Edge. 2015. A seccomp overview. `https://lwn.net/Articles/656307/`. (2015).

[27] Matthew Edman and Paul Syverson. 2009. AS-awareness in Tor Path Selection. In *ACM Conference on Computer and Communications Security (CCS).*

[28] Enter the Andromeda zone - Google Cloud Platform latest networking stack [n. d.]. Enter the Andromeda zone - Google Cloud Platform latest networking stack. `http://goo.gl/u59Iw1`. ([n. d.]).

[29] ETSI Network Function Virtualization [n. d.]. ETSI Network Function Virtualization. `http://www.etsi.org/technologies-clusters/technologies/nfv`. ([n. d.]).

[30] Nathan S Evans, Roger Dingledine, and Christian Grothoff. 2009. A Practical Congestion Attack on Tor Using Long Paths. In *USENIX Security Symposium.*

[31] Christina Garman, Matthew Green, and Ian Miers. 2014. Decentralized anonymous credentials. In *Network and Distributed System Security Symposium (NDSS).*

[32] Aaron Gember, Prathmesh Prabhu, Zainab Ghadiyali, and Aditya Akella. 2012. Toward software-defined middlebox networking. In *Workshop on Hot Topics in Networks (HotNets).*

[33] Yossi Gilad and Amir Herzberg. 2012. Spying in the dark: TCP and Tor traffic analysis. In *Privacy Enhancing Technologies Symposium (PETS).*

[34] Stephen Herwig, Christina Garman, and Dave Levin. 2020. Achieving Keyless CDNs with Conclaves. In *USENIX Security Symposium.*

[35] Matthew Hoekstra, Reshma Lal, Pradeep Pappachan, Vinay Phegade, and Juan Del Cuvillo. 2013. Using innovative instructions to create trustworthy software solutions. In *International Workshop on Hardware and Architectural Support for Security and Privacy (HASP).*

[36] Nicholas Hopper, Eugene Y Vasserman, and Eric Chan-Tin. 2010. How much anonymity does network latency leak? *ACM Transactions on Information and System Security (TISSEC)* 13, 2 (2010), 13.

[37] Intel. [n. d.]. Code Sample: Intel® Software Guard Extensions Remote Attestation End-to-End Example. `https://software.intel.com/content/www/us/en/develop/articles/code-sample-intel-software-guard-extensions-remote-attestation-end-to-end-example.html`. ([n. d.]).

[38] Ankur Jain, Joseph M. Hellerstein, Sylvia Ratnasamy, and David Wetherall. 2004. A Wakeup Call for Internet Monitoring Systems: The Case for Distributed Triggers. In *Workshop on Hot Topics in Networks (HotNets).*

[39] Xin Jin, Erran Li Li, Laurent Vanbever, and Jennifer Rexford. 2013. SoftCell: Scalable and flexible cellular core network architecture. In *ACM Conference on emerging Networking EXperiments and Technologies (CoNEXT).*

[40] Aaron Johnson, Rob Jansen, Nicholas Hopper, Aaron Segal, and Paul Syverson. 2017. PeerFlow: Secure load balancing in Tor. In *Privacy Enhancing Technologies Symposium (PETS).*

[41] Aaron Johnson, Chris Wacek, Rob Jansen, Micah Sherr, and Paul Syverson. 2013. Users get routed: Traffic correlation on Tor by realistic adversaries. In *ACM Conference on Computer and Communications Security (CCS).*

[42] Simon Johnson, Vinnie Scarlata, Carlos Rozas, Ernie Brickell, and Frank Mckeen. [n. d.]. ([n. d.]).

[43] Dilip Joseph and Ion Stoica. 2008. Modeling Middleboxes. *IEEE Network* 22, 5 (2008), 20–25.

[44] Dilip A. Joseph, Arsalan Tavakoli, and Ion Stoica. 2008. A Policy-aware Switching Layer for Data Centers. In *ACM SIGCOMM.*

[45] Marc Juarez, Mohsen Imani, Mike Perry, and Claudia Diaz. 2016. Toward an Efficient Website Fingerprinting Defense. In *European Symposium on Research in Computer Security (ESORICS).*

[46] Sheharbano Khattak, Mobin Javed, Philip D. Anderson, and Vern Paxson. 2013. Towards Illuminating a Censorship Monitor's Model to Facilitate Evasion. In *USENIX Workshop on Free and Open Communications on the Internet (FOCI).*

[47] Hyojoon Kim, Joshua Reich, Arpit Gupta, Muhammad Shahbaz, Nick Feamster, and Russ Clark. 2015. Kinetic: Verifiable Dynamic Network Control. In *Symposium on Networked Systems Design and Implementation (NSDI).*

[48] Seong Min Kim, Juhyeng Han, Jaehyeong Ha, Taesoo Kim, and Dongsu Han. 2017. Enhancing Security and Privacy of Tor's Ecosystem by Using Trusted Execution Environments. In *Symposium on Networked Systems Design and Implementation (NSDI).*

[49] Katharina Kohls, Kai Jansen, David Rupprecht, Thorsten Holz, and Christina Pöpper. 2019. On the Challenges of Geographical Avoidance for Tor. In *Network and Distributed System Security Symposium (NDSS).*

[50] Albert Kwon, Mashael AlSabah, David Lazar, Marc Dacier, and Srinivas Devadas. 2015. Circuit Fingerprinting Attacks: Passive Deanonymization of Tor Hidden Services. In *USENIX Annual Technical Conference.*

[51] Dave Levin, Youndo Lee, Luke Valenta, Zhihao Li, Victoria Lai, Cristian Lumezanu, Neil Spring, and Bobby Bhattacharjee. 2015. Alibi Routing. In *ACM SIGCOMM.*

[52] Zhihao Li, Stephen Herwig, and Dave Levin. 2017. DeTor: Provably Avoiding Geographic Regions in Tor. In *USENIX Security Symposium.*

[53] D.J.C. MacKay. 2005. Fountain codes. *IEEE Proceedings-Communications* 152, 6 (Dec. 2005).

[54] Frank McKeen, Ilya Alexandrovich, Alex Berenzon, Carlos V. Rozas, Hisham Shafi, Vedvyas Shanbhogue, and Uday R. Savagaonkar. 2013. Innovative instructions and software model for isolated execution. In *International Workshop on Hardware and Architectural Support for Security and Privacy (HASP).*

[55] Stephen J. Murdoch and George Danezis. 2005. Low-cost traffic analysis of Tor. In *USENIX Security Symposium.*

[56] Steven J. Murdoch and Piotr Zieklński. 2007. Sampled Traffic Analysis by Internet-Exchange-Level Adversaries. In *International Workshop on Privacy Enhancing Technologies.*

[57] David Naylor, Kyle Schomp, Matteo Varvello, Ilias Leontiadis, Jeremy Blackburn, Diego Lopez, Konstantina Papagiannaki, Pablo Rodriguez Rodriguez, and Peter Steenkiste. 2015. Multi-Context TLS (mcTLS): Enabling Secure In-Network Functionality in TLS. In *ACM SIGCOMM.*

[58] ninequestions [n. d.]. Nine Questions about Hidden Services. `https://blog.torproject.org/nine-questions-about-hidden-services`. ([n. d.]).

[59] onionbalance [n. d.]. OnionBalance. https://onionbalance.readthedocs.io/en/latest/. ([n. d.]).

[60] Lasse Øverlier and Paul Syverson. 2006. Valet Services: Improving Hidden Servers with a Personal Touch. In *PETS.*

[61] Andriy Panchenko and Johannes Renner. 2009. Path Selection Metrics for Performance-Improved Onion Routing. In *Symposium on Applications and the Internet (SAINT).*

[62] Rishabh Poddar, Chang Lan, Raluca Ada Popa, and Sylvia Ratnasamy. 2018. SafeBricks: Shielding Network Functions in the Cloud. In *Symposium on Networked Systems Design and Implementation (NSDI).*

[63] Matthew Prince. [n. d.]. The Trouble with Tor. ([n. d.]). `https://blog.cloudflare.com/the-trouble-with-tor/`.

[64] prop305 [n. d.]. Tor spec: ESTABLISH_INTRO Cell DoS Defense Extension. `https://gitweb.torproject.org/torspec.git/tree/proposals/305-establish-intro-dos-defense-extention.txt`. ([n. d.]).

[65] Reporters Without Borders. 2013. Enemies of the Internet 2013, Report. `http://surveillance.rsf.org/en/wp-content/uploads/sites/2/2013/03/enemies-of-the-internet_2013.pdf`. (March 2013).

[66] Florentin Rochet, Olivier Bonaventure, and Olivier Pereira. 2019. Flexible Anonymous Network. In *Workshop on Hot Topics in Privacy Enhancing Technologies (HotPETs).*

[67] Max Schuchard, John Geddes, Christopher Thompson, and Nicholas Hopper. 2012. Routing Around Decoys. In *ACM Conference on Computer and Communications Security (CCS).*

[68] sgx-patch [n. d.]. L1 Terminal Fault. `https://software.intel.com/security-software-guidance/software-guidance/l1-terminal-fault`. ([n. d.]).

[69] Micah Sherr, Matt Blaze, and Boon Thau Loo. 2009. Scalable Link-based Relay Selection for Anonymous Routing. In *Privacy Enhancing Technologies Symposium (PETS).*

[70] Justine Sherry, Shaddi Hasan, Colin Scott, Arvind Krishnamurthy, Sylvia Ratnasamy, and Vyas Sekar. 2012. Making Middleboxes Someone Else's Problem: Network Processing as a Cloud Service. In *ACM SIGCOMM.*

[71] Justine Sherry, Chang Lan, Raluca Ada Popa, and Sylvia Ratnasamy. 2015. Blind-Box: Deep Packet Inspection over Encrypted Traffic. In *ACM SIGCOMM.*

[72] Seungwon Shin, Phillip A. Porras, Vinod Yegneswaran, Martin W. Fong, Guofei Gu, and Mabry Tyson. 2013. FRESCO: Modular Composable Security Services for Software-Defined Networks.. In *Network and Distributed System Security Symposium (NDSS).*

[73] Payap Sirinam, Mohsen Imani, Marc Juarez, and Matthew Wright. 2018. Deep Fingerprinting: Undermining Website Fingerprinting Defenses with Deep Learning. In *ACM Conference on Computer and Communications Security (CCS).*

[74] R. Soule, S. Basu, P. J. Marandi, F. Pedone, R. Kleinberg, E. G.Sirer, and N. Foster. 2014. Merlin:A Language for Provisioning Network Resources. In *ACM Conference on emerging Networking EXperiments and Technologies (CoNEXT).*

[75] Stem Controller Library. [n. d.]. `https://stem.torproject.org`. ([n. d.]).

[76] Ceysun Sucu. 2015. Tor: Hidden Service Scaling. University College London MSc Thesis. (2015).

[77] Yixin Sun, Anne Edmundson, Laurent Vanbever, Oscar Li, Jennifer Rexford, Mung Chiang, and Prateek Mittal. 2015. RAPTOR: Routing Attacks on Privacy in Tor. In *USENIX Security Symposium.*

[78] Tor: Hidden Service Protocol [n. d.]. Tor: Hidden Service Protocol. https://www.torproject.org/docs/hidden-services.html.en. ([n. d.]).
[79] Bohdan Trach, Alfred Krohmer, Franz Gregor, Sergei Arnautov, Pramod Bhatotia, and Christof Fetzer. 2018. ShieldBox: Secure Middleboxes using Shielded Execution. In *Symposium on SDR Research (SOSR)*.
[80] Chia-Che Tsai, Donald E. Porter, and Mona Vij. 2017. Graphene-SGX: A Practical Library OS for Unmodified Applications on SGX. In *USENIX Annual Technical Conference*.
[81] Jo Van Bulck, Marina Minkin, Ofir Weisse, Daniel Genkin, Baris Kasikci, Frank Piessens, Mark Silberstein, Thomas F Wenisch, Yuval Yarom, and Raoul Strackx. 2018. Foreshadow: Extracting the Keys to the Intel SGX Kingdom with Transient Out-of-Order Execution. In *USENIX Security Symposium*.
[82] Tao Wang, Xiang Cai, Rishab Nithyanand, Rob Johnson, and Ian Goldberg. 2014. Effective Attacks and Provable Defenses for Website Fingerprinting. In *USENIX Security Symposium*.
[83] Tao Wang and Ian Goldberg. 2013. Improved Website Fingerprinting in Tor. In *Workshop on Privacy in the Electronic Society (WPES)*.
[84] Tao Wang and Ian Goldberg. 2017. Walkie-Talkie: An Efficient Defense Against Passive Website Fingerprinting Attacks. In *USENIX Security Symposium*.
[85] Zhaoguang Wang, Zhiyun Qian, Qiang Xu, Zhuoqing Mao, and Ming Zhang. 2011. An untold story of middleboxes in cellular networks. In *ACM SIGCOMM*.
[86] Philipp Winter and Stefan Lindskog. 2012. How the Great Firewall of China is Blocking Tor. In *USENIX Workshop on Free and Open Communications on the Internet (FOCI)*.
[87] Lei Yang and Fengjun Li. 2015. mTor: A multipath Tor routing beyond bandwidth throttling. In *IEEE Conference on Communications and Network Security (CNS)*.
[88] Xiaowei Yang, David Wetherall, and Thomas Anderson. 2008. TVA: A DoS-limiting network architecture. *Networking, IEEE/ACM Transactions on* 16, 6 (2008), 1267–1280.

Appendices are supporting material that has not been peer-reviewed.

A EXAMPLE CODE

This appendix contains sample code of the `Browser` function. This and other example functions are provided in our public code repository, available at `https://bento.cs.umd.edu`.

```
def browser(url, padding):
    # Fetch contents of site
    body = requests.get(url, timeout=1).content

    # Compress contents
    compressed = zlib.compress(body)

    # Pad to nearest multiple of 'padding'
    final = compressed
    if padding - len(final) > 0:
        final = final +
         (os.urandom(padding - len(final)))
    else:
        final = final +
         (os.urandom((len(final) + padding) % padding))

    api.send(final)
```

Listing 1: Python implementation of `browser` function.

B ARTIFACT APPENDIX

Abstract

Our certified artifact consists of prototype implementations for the Bento server, as well as clients and functions that showcase various use cases. The server may run client functions either in normal userland or within an SGX enclave, based on command-line arguments.

Scope

The provided artifact allows one to setup and run their own Bento server and clients. As such, one can validate all experiments from Sections 7 and 8. The software specific to each evaluation is located in the *experiments* directory. Each evaluation has a corresponding README:

- Browser – upload and execute a function that fetches a webpage, padding the response with dummy bytes.
- Cover – upload and execute a function that fetches a webpage, generating coverage traffic in the process.
- LoadBalancer – run a function that acts as a load balancer for a hidden service.
- WebsiteFingerprinting – The deep fingerprinting attack [73]; used to evaluate the effectiveness of Browser's padding scheme.

Additionally, once they have setup a client and server, a user can also develop and write their own Bento functions!

Contents

Our artifact contains four primary pieces: 1) the source for the client and server packages, 2) a set of self-contained projects and experiments using Bento (these correspond to the experiments in the paper), 3) a set of simple tests and sample functions, designed to get the user started and introduce them to both running functions

and writing their own, 4) thorough documentation, including the source for the Bento website, which contains detailed documentation for setting up and running a client and server, as well as the various functions described in the paper.

Hosting

Our verified code can be found at `https://github.com/breakerspace/bento`, commit 1de0d46.

Requirements

We developed and successfully tested our Bento client and server code on Ubuntu 18.04 with Python 3.6. Our development machine was an Intel NUC (NUC10i7FNH).

Our Bento server optionally requires access to SGX-capable hardware. For running functions within an SGX enclave, we use the Graphene-SGX LibOS available at `https://github.com/oscarlab/graphene`, version 1.1, commit 73b774f. Additionally, we use the SGX Linux kernel module from `https://github.com/intel/linux-sgx-driver`, version 2.11.0, commit 2d2b795, and the SGX platform software (that is, the SDK and PSW) from `https://github.com/intel/linux-sgx`, version 2.9.1, commit fdc9b33.