



237.17

Рейтинг

## Digital Security

Безопасность как искусство



kerbyj 13 сентября в 07:01

# Что мы используем для анализа Android-приложений

Блог компании Digital Security, Информационная безопасность\*, Разработка под Android\*, Тестирование мобильных приложений\*

Всем привет! В этой статье расскажем про инструментарий для анализа мобильных приложений, который мы используем каждый день. Для начала поговорим про то, как запускать мобильные приложения, чем смотреть трафик, а также рассмотрим инструменты для статического и динамического анализа мобильных приложений.

## Девайсы

### Настоящее устройство

В Digital Security примерно 99% аудитов проводятся на настоящих девайсах. Это избавляет от множества проблем. С ходу могу назвать следующие проблемы, которые решает использование реального устройства:

- Проще пройти аттестацию среды, если таковая реализована в приложении
- Не тратятся полезные ресурсы на рабочей машине
- Встречаются приложения, у которых некоторые функциональности вынесены в нативные библиотеки, и при этом они скомпилированы только для архитектуры ARM. Большинство виртуальных девайсов — это x86\_64 или x64; в Android Studio есть ARM-девайс, но версия Android и быстродействие оставляют желать лучшего
- Быстродействие эмулятора всегда ниже, чем у реального устройства

В основном мы используем Samsung с Android 12, пару Xiaomi и пикселей. Также недавно у нас появился планшет на Harmony OS.

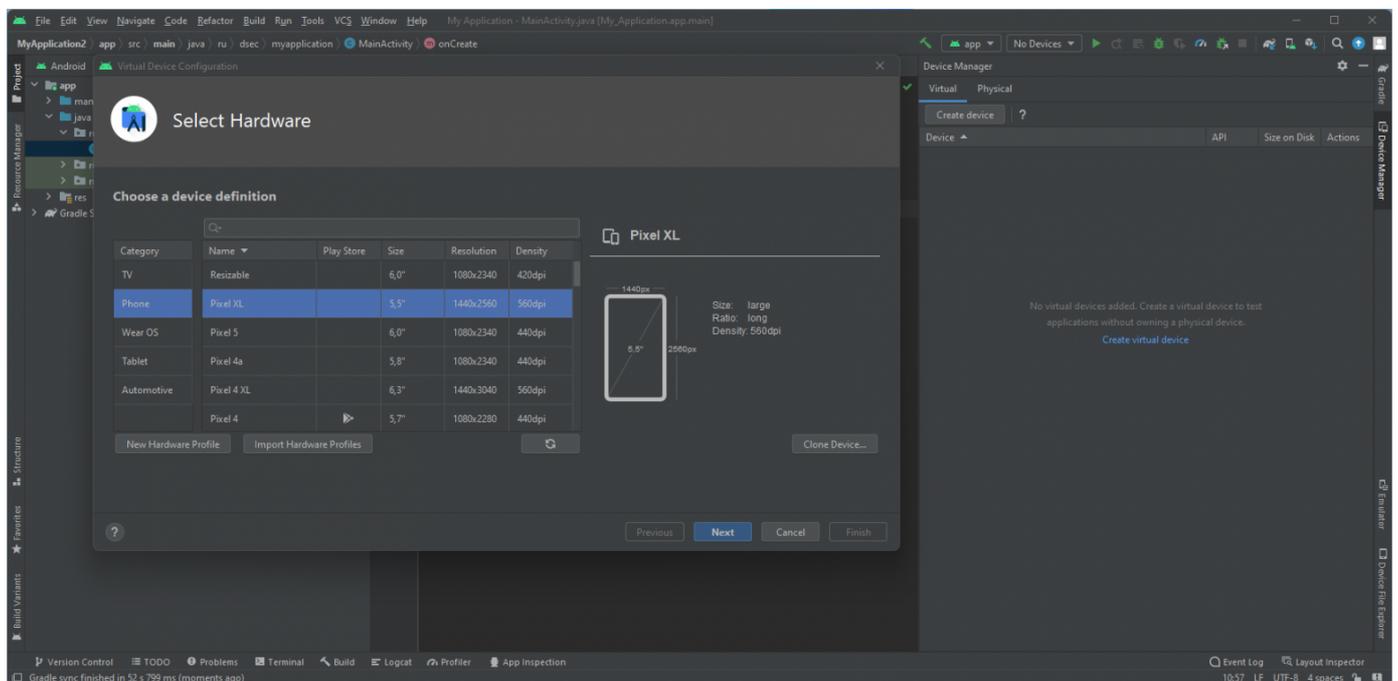
### Виртуальный девайс

## Android Studio Emulator

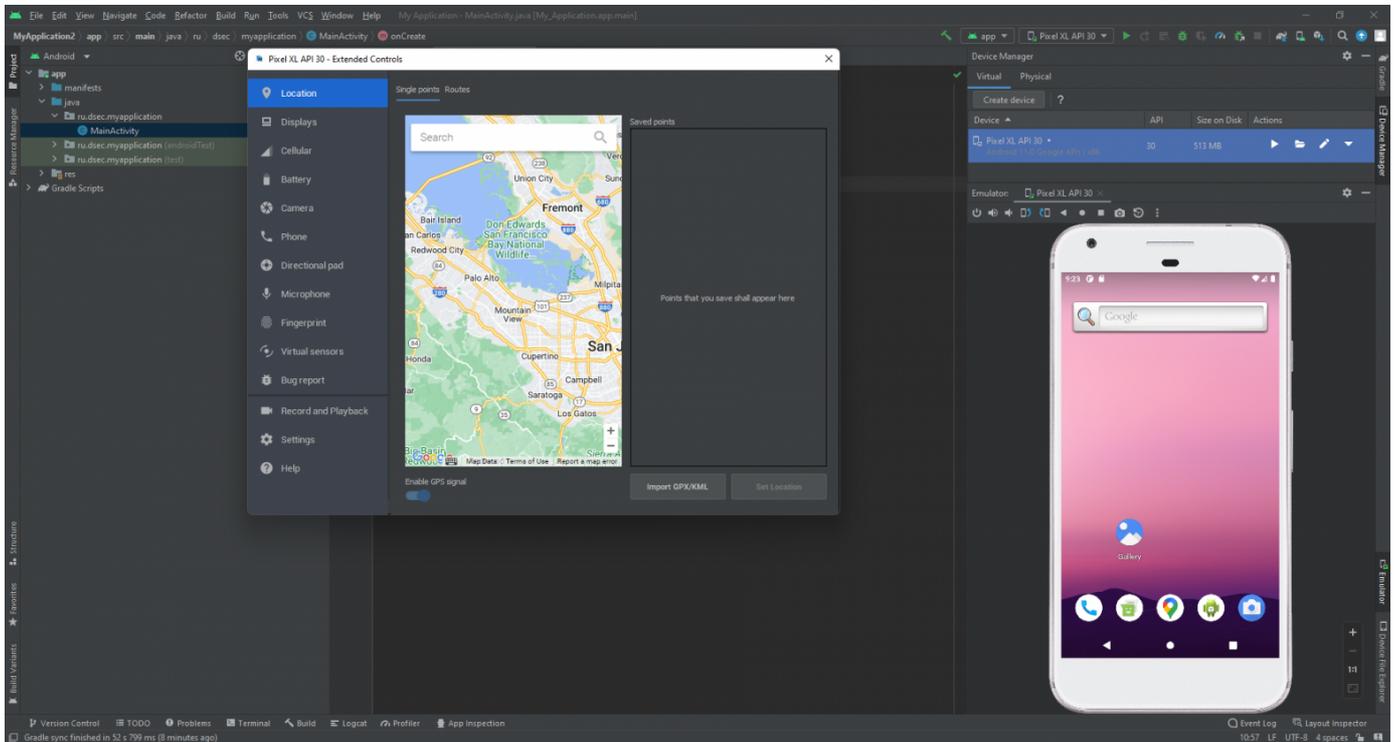


Чтобы создать виртуальное устройство, создаем пустой проект в Android Studio, выбираем вкладку Tools, где нам нужен инструмент Device Manager. Откроется боковая вкладка с девайсами.

На следующем шаге мы можем выбрать из двух версий — с Google Play и без. Если выбрать первую, то у вас не будет возможности выйти в режим с правами суперпользователя.



После скачивания необходимых файлов на вкладке Device Manager появится ваше устройство. Можем запускать.



## Windows Subsystem for Android

Про этот интересный инструмент мы уже рассказали в [прошлой статье по настройке WSA](#) — обязательно посмотрите :)

## Прокси

Анализ мобильных приложений неразрывно связан с анализом трафика между сервером и клиентом (мобильным приложением). Для этого можно найти множество (нет) инструментов. Но у нас в основном встречается три. О них ниже.

## BurpSuite

<https://portswigger.net/burp/releases/>

Это наш главный инструмент, который запущен почти всегда. Как бы это странно ни звучало, но у него самый понятный интерфейс и есть все, что нужно, из коробки. Распространяется в двух версиях — Community и Professional. Для базового анализа хватит и первой версии, к тому же всегда можно добавить нужные расширения через Extender.

## MitmProxy



# mitmproxy

<https://mitmproxy.org/>

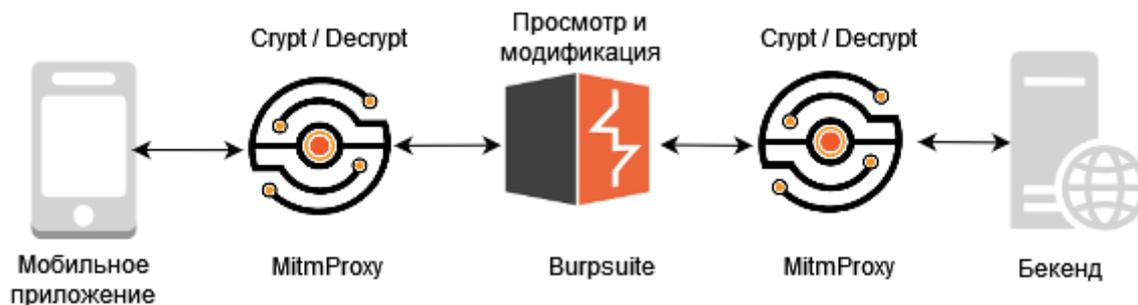
Консольная тулза, которая позволяет, как и Burp, завернуть в нее трафик и изучать его. Есть еще веб-интерфейс, но более интересна другая функциональность.

```
~/mitmproxy/mitmproxy
Flows
GET https://www.google.com/
  ← 200 text/html 64.52k 487ms
GET https://www.google.com/logos/doodles/2018/doodle-snow-games-day-12-6070619765473280-s.png
  ← 200 image/png 2.63k 184ms
GET https://www.google.com/logos/2018/snowgames_skijump/cta.png
  ← 200 image/png 13.4k 229ms
>> GET https://www.gstatic.com/external_hosted/createjs/createjs-2015.11.26.min.js
  ← 200 text/javascript 48.51k 475ms
GET https://ssl.gstatic.com/gb/images/i2_2ec824b0.png
  ← 200 image/png 23.64k 253ms
GET https://ssl.gstatic.com/safebrowsing/csd/client_model_v5_variation_0.pb
  ← 200 application/octet-stream 67.92k 356ms
GET https://ssl.gstatic.com/safebrowsing/csd/client_model_v5_ext_variation_0.pb
  ← 200 application/octet-stream 67.92k 412ms
GET https://www.google.com/logos/2018/snowgames_skijump/snowgames_skijump18.js
  ← 200 text/javascript 258.16k 900ms
POST https://www.google.com/gen_204?s=webaft&atyp=csi&ei=vCGLWr6uMsKk0gTYs6yIAw&rt=wsrt.2615,aft.1379,prt.1379
  ← 204 text/html [no content] 379ms
GET https://www.gstatic.com/og/_/js/k=og.og2.en_US.u1Hn0gN1l6I.0/rt=j/m=def/exm=in,fot/d=1/ed=1/rs=AA2YrT
uVOKajN...
  ← 200 text/javascript 46.4k 265ms
GET https://www.google.com/xjs/_/js/k=xjs.s.en.zjivxe8fVgY.0/m=sx,sb,cdos,cr,eLog,hsm,jsa,r,d,csi/am=wCL0
eMEByP8...
  ← 200 text/javascript 144.26k 368ms
GET https://www.google.com/xjs/_/js/k=xjs.s.en.zjivxe8fVgY.0/m=aa,abd,async,dvl,foot,fpe,ipv6,lu,m,mu,sf,
sonic,s...
  ← 200 text/javascript 30.54k 195ms
GET https://www.google.com/logos/2018/snowgames_skijump/main-sprite.png
  ← 200 image/png 13.4k 229ms
↓ [14/36] [*:9999]
: replay.client [fLow]
```

MitmProxy позволяет легко дописывать функционал при помощи Python API. Например, у нас был кейс, когда было реализовано шифрование данных, пересылаемых через вебсокеты, и хотелось посмотреть на них. А еще больше хотелось их модифицировать.

Буквально за несколько минут был написан скрипт, который расшифровывал данные (у нас были необходимые ключи), передавал их дальше в Burp, который был указан как upstream proxy, и был еще один upstream proxy, который зашифровывал данные и передавал уже на бэкэнд.

Получился примерно такой флоу:



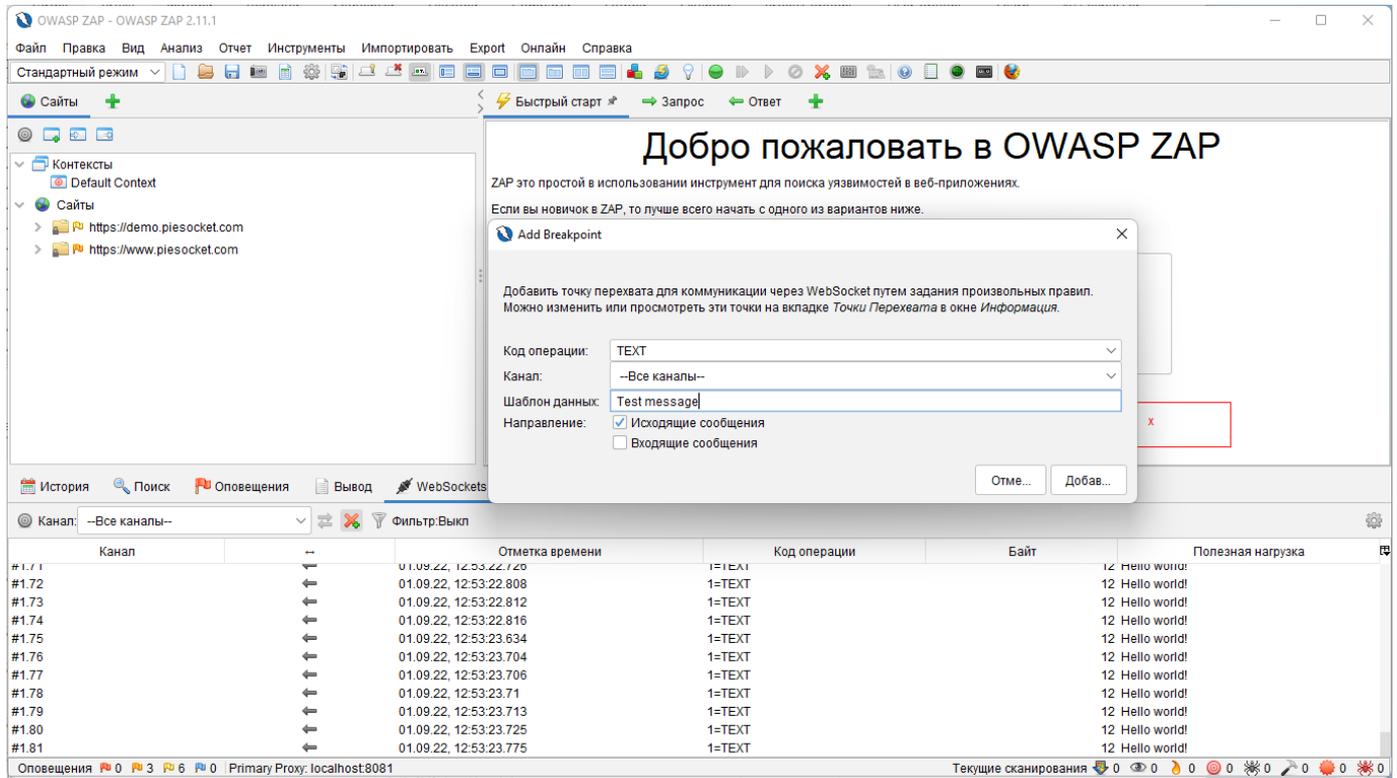
## OWASP ZAP

<https://www.zaproxy.org/download/>

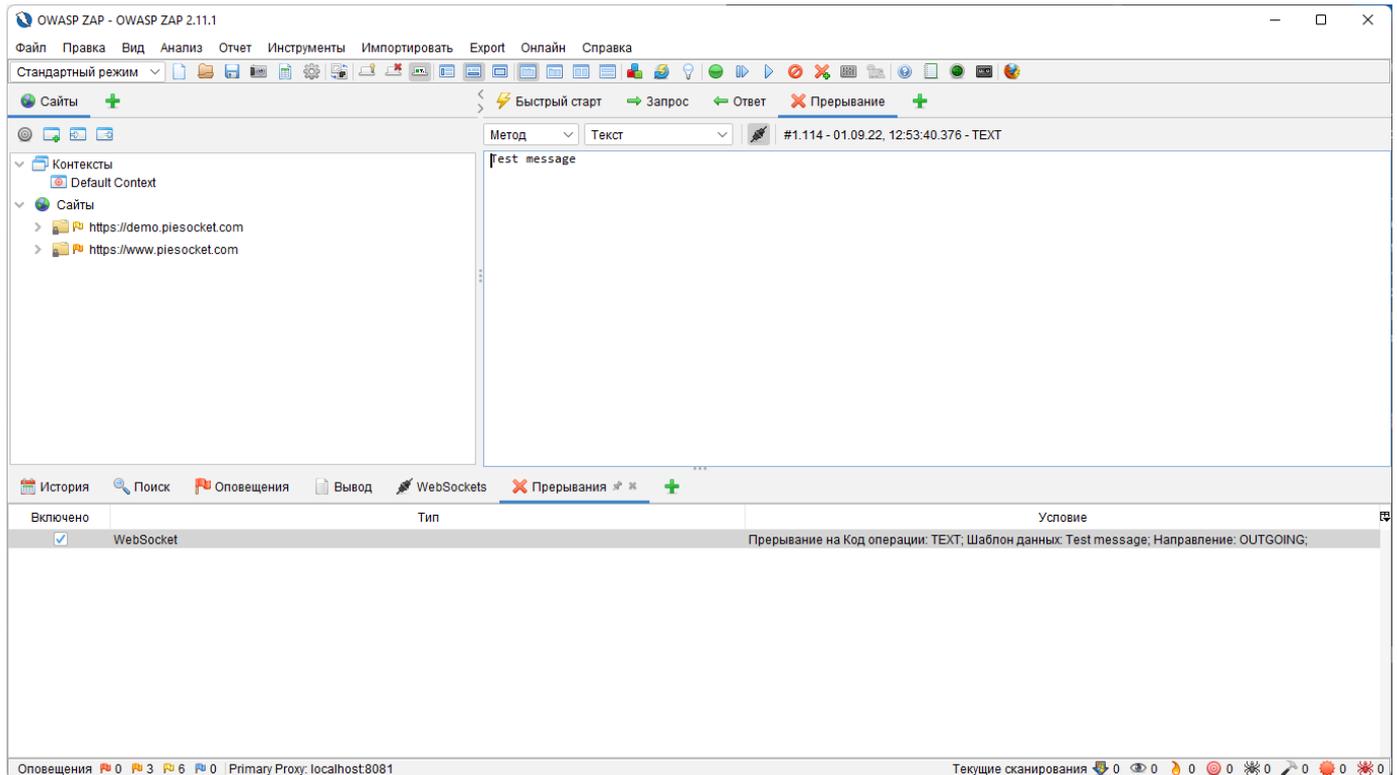
Еще один мощный инструмент, который практически не уступает BurpSuite. Единственный его минус — миллиард настроек, в которых можно бесконечно искать необходимую галку.

Хочется выделить одну из функций, которой нет в BurpSuite и которая при этом иногда необходима — возможность ставить точку остановки для вебсокетов. В современном мире встречаются сайты или приложения, у которых почти все клиент-серверное взаимодействие основано на вебсокетах. В BurpSuite вы можете включить Interceptor и прокликать каждое сообщение, пока не появится то самое, нужное вам, которое вы хотите модифицировать и отправить дальше. Но что делать, если таких сообщений десятки или сотни? Здесь и пригодится этот функционал.

Добавить брейкпойнт можно на вкладке с вебсокетами. Мы определяем тип передаваемых данных и данные для поиска.



Как только появится сообщение, которое удовлетворяет нашему шаблону, произойдет прерывание, а ZAP покажет искомое сообщение.

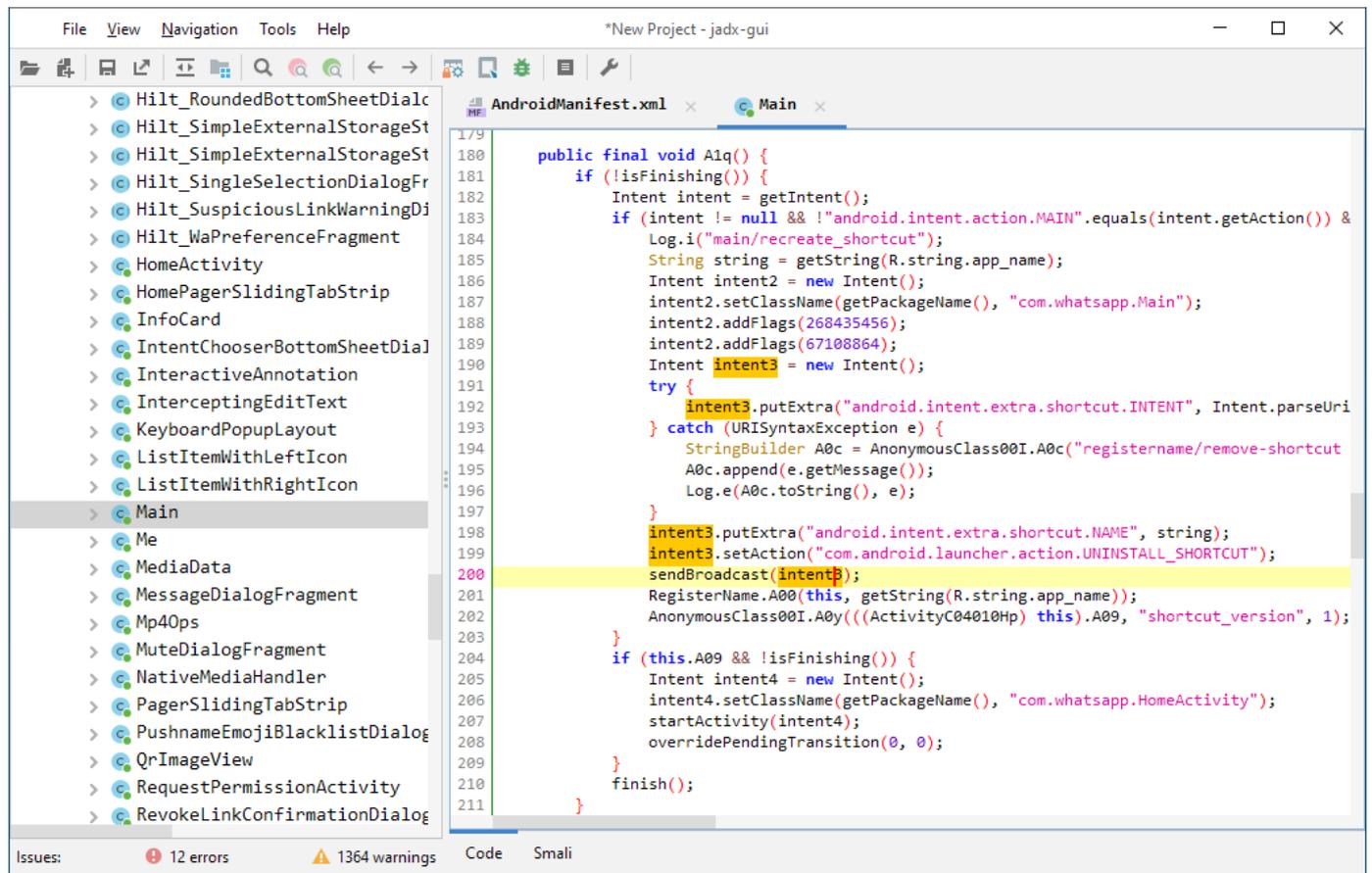


## Статический анализ

### JADX

<https://github.com/skylot/jadx>

Инструмент, с которым знакомы почти все аналитики безопасности мобильных приложений и не только. Это самый настоящий мастхэв при анализе приложений. Если при изучении трафика мы 70% времени проводим в BurpSuite, то при исследовании мобильного приложения «черным ящиком» 80% времени открыт JADX.



Позволяет на лету декомпилировать код, просматривать классы, ресурсы и делать поиск по всему приложению. Самый настоящий комбайн в хорошем смысле этого слова.

Также позволяет подключиться к приложению в режиме отладки и ставить брейкпойнты.

## Apktool

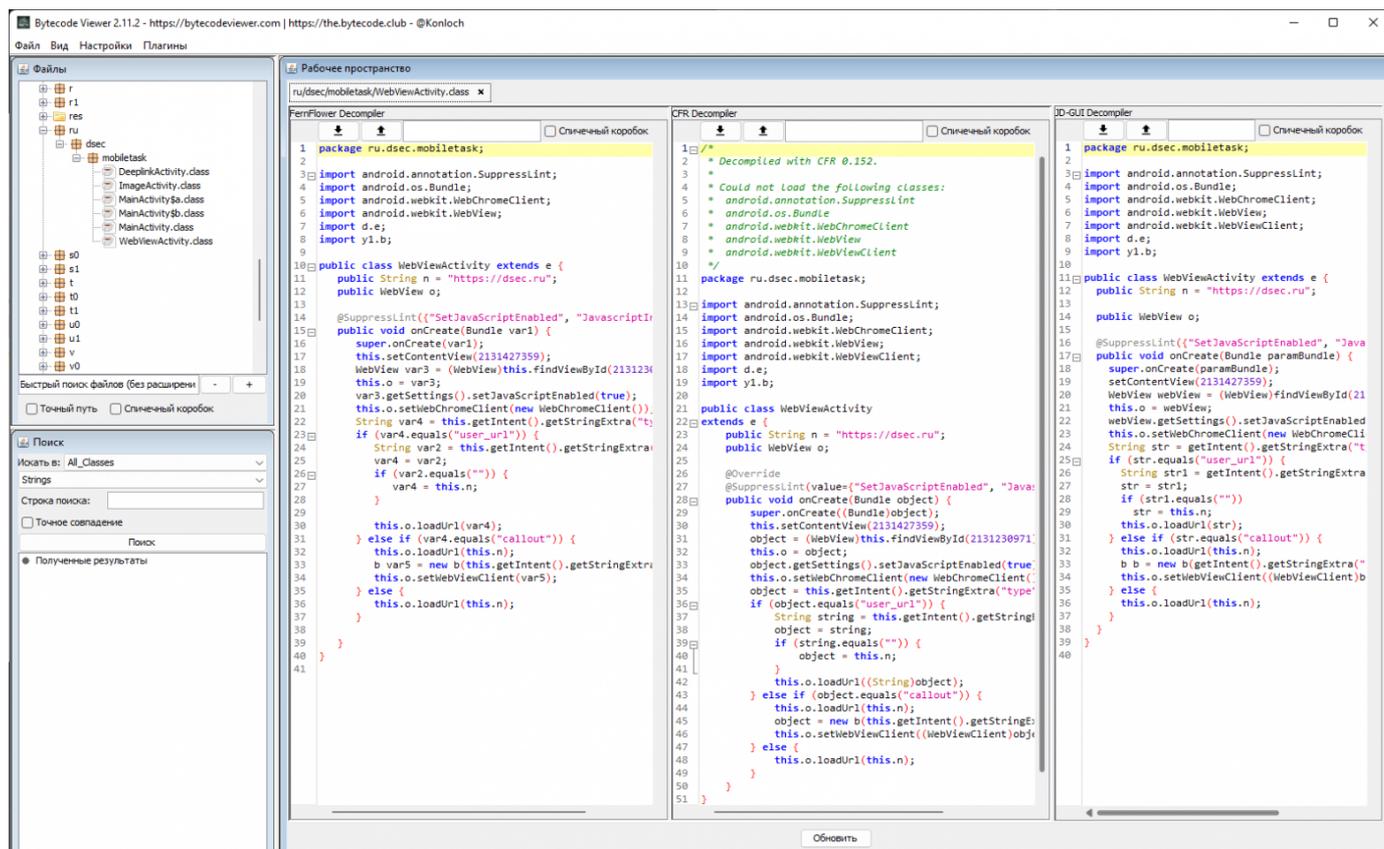
<https://ibotpeaches.github.io/Apktool/>

Используется ровно в двух случаях — когда нужно достать используемые ресурсы или ассеты, а также когда JADX не хватает памяти для того, чтобы открыть и декомпилировать все приложение, чтобы поискать по нему. В таком случае распаковываем приложение при помощи apktool, делаем поиск обычным grep/ripgrep и открываем класс либо в jadx, либо файл со smali-кодом в любом текстовом редакторе.

## ByteCode Viewer

<https://github.com/Konloch/bytecode-viewer>

All-In-One тулза для анализа мобильных приложений. Можно закинуть APK и сравнить результаты сразу нескольких декомпиляторов. Очень полезно, когда JADX не может привести некоторые методы к нормальному виду, а через smali ничего не понятно :)

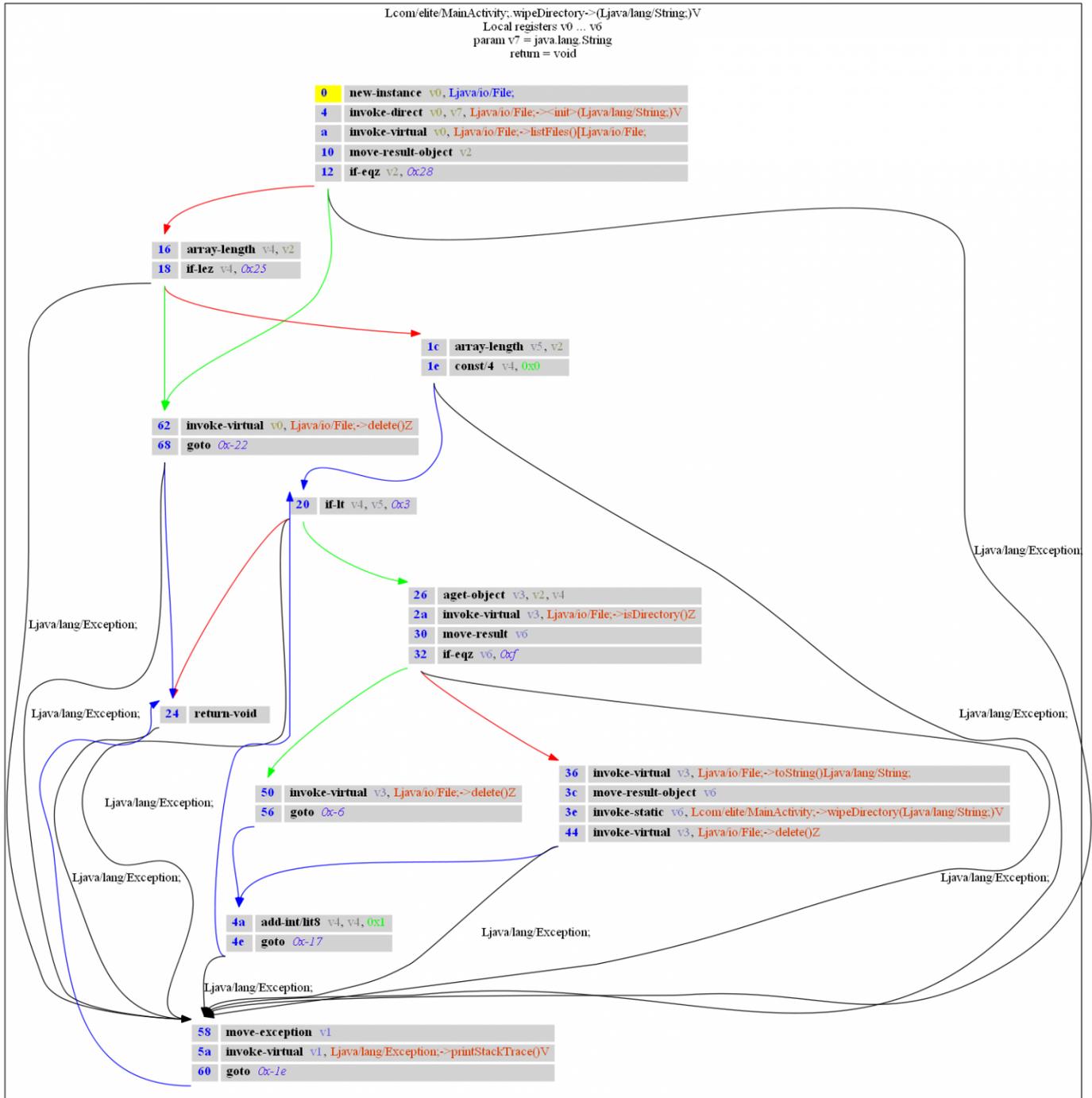


## Androguard

<https://androguard.readthedocs.io/en/latest/>

Очень мощный инструмент для реверс-инжиниринга мобильных приложений. Несколько раз использовали его для автоматизации проверок, когда необходимо было пройтись по функциям и XREF'ам.

Также умеет рисовать графы для классов, что визуально может помочь разобраться в флоу работы мобильного приложения.



## Mariana Trench



<https://github.com/facebook/mariana-trench>

Один из наиболее "свежих" инструментов для анализа мобильных приложений. Производит Taint-анализ по байткоду Dalvik. Легко устанавливается, из коробки уже имеет некоторый набор проверок, что поможет найти низко висящие уязвимости. Естественно, под каждый проект лучше писать свои правила — это повысит эффект от анализа и позволит сэкономить время.

Главное — учесть, что если у вас приложение большое и имеет кучу функционала, то нужно запастись временем и терпением — анализ может занять до получаса. Еще из замеченного: `mariana trench` не учитывает, экспортируемый ли компонент.

Представим базовую уязвимость, когда у нас в активити есть обработка диплинка, из него берется параметр `url` и передается в функцию `loadUrl` какой-нибудь `webView`. Mariana Trench подсветит это как уязвимость, однако если точка входа (Activity) у нас неэкспортируемая, то это будет False Positive.

Хоть это и стоит иметь в виду, в отчет такое не напишешь.

## Hbctool

<https://github.com/bongtrop/hbctool>

<https://suam.wtf/posts/react-native-application-static-analysis-en/>

Консольная утилита для работы с байткодом Hermes. Одним из популярных фреймворков для кроссплатформенной разработки является React Native. Когда вы анализируете такое приложение, у вас есть три варианта развития событий:

- В файле `./assets/index.android.bundle` будет минифицированный javascript-код, который можно попробовать прогнать через `beautifier` и прочесть.
- Рядом с файлом `./assets/index.android.bundle` будет лежать файл `./assets/index.android.bundle.map` с маппингами для собранного файла, и вы можете при помощи <https://github.com/rarecoil/unwebpack-sourcemap> распаковать его и прочесть оригинальный исходный код.
- В файле `./assets/index.android.bundle` будет бинарщина. Это и есть Hermes Bytecode.

Эта тулза на крайний случай. При помощи нее можно попробовать превратить код в так называемый NASM. Читать его все еще трудно, но это явно лучше, чем бинарный файл.

И через кровь и слезы можно попытаться изучить логику работы, изменить и собрать

обратно.

## Динамический анализ

### Frida

# FRIDA

<https://frida.re/>

Самый известный инструмент, который позволяет на лету подключаться к приложению и внедрять в него свой код. Если сравнивать с чем-то по назначению, то в голову приходит только Xposed Framework. Если сравнивать по времени, то написание скрипта для Frida занимает намного меньше времени, чем модуля для Xposed.

Чтобы установить необходимые вещи на хост, нужно выполнить следующую простую команду:

```
pip install frida-tools
```

Скачиваем `frida-server` для нужной архитектуры, переносим его на устройство и запускаем с правами суперпользователя:

```
su  
cd /data/local/tmp  
/data/local/tmp/frida-server-15.2.2-android-arm64
```

После этого мы сможем увидеть его в консоли и начать писать скрипты для приложений.

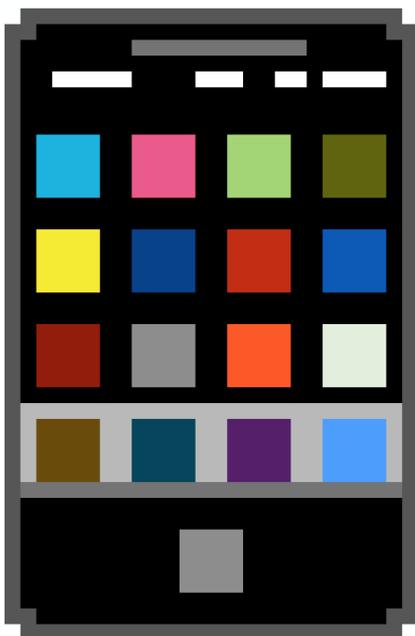
```
C:\Users\kerby\AppData\Roam > Windows PowerShell
DDV_sprout:/ # cd /data/local/tmp
DDV_sprout:/data/local/tmp # whoami
root
DDV_sprout:/data/local/tmp # ./frida-server-15.1.22-android-arm64
```

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\kerby> frida-ls-devices.exe
Id          Type      Name
-----
Local      Local    Local System
DBAA862788JB6817329  usb      Nokia 7 2
socket     remote   Local Socket
PS C:\Users\kerby> |
```

## Objection



OBJECTION  
RUNTIME  
MOBILE  
EXPLORATION  
GIT.IO/OBJECTION

<https://github.com/sensepost/objection>

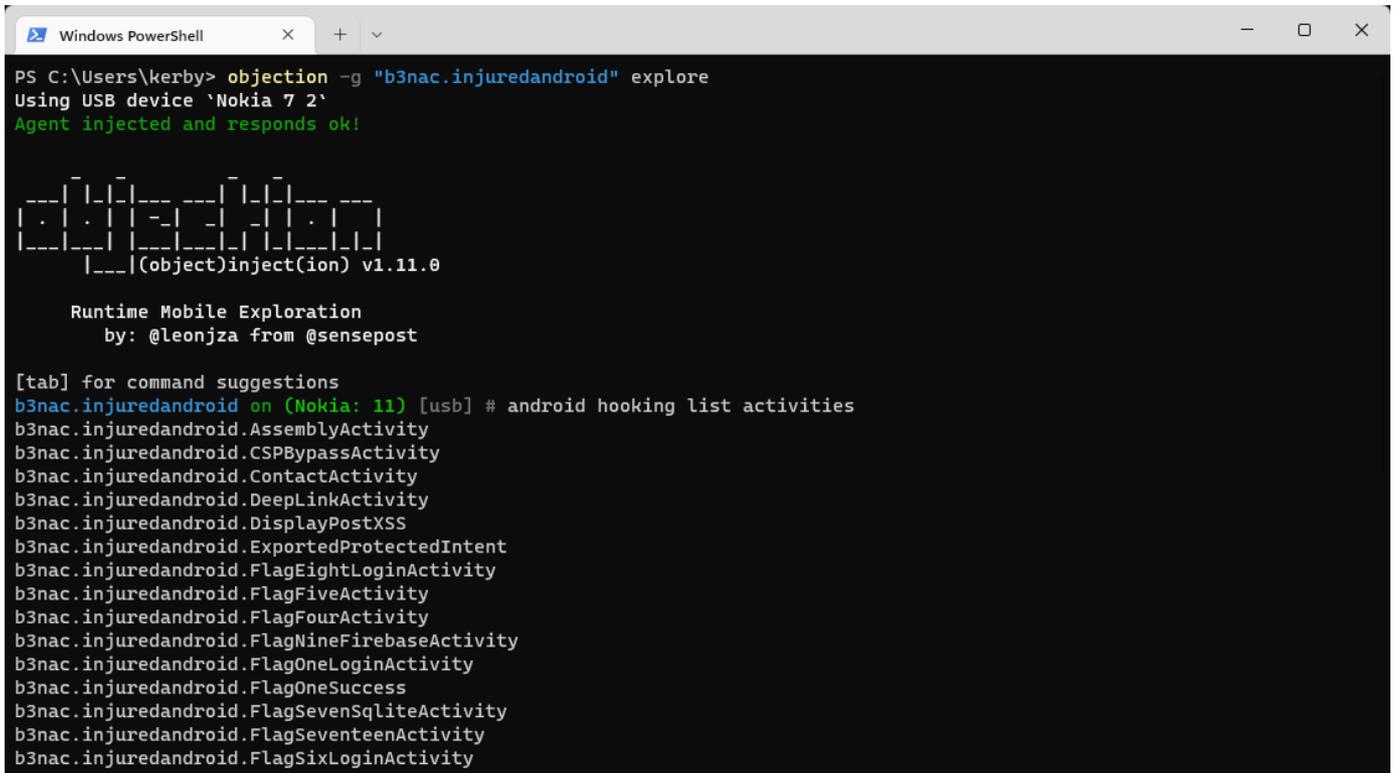
Objection — это тулkit, который основан на Frida и предоставляет уже готовые функции для работы с мобильными приложениями. Имеет следующие готовые функции:

- Работа с файловой системой

- Байпасс SSL-pinning
- Дамп keychain & keystore
- Работа с памятью
- Манипуляции с интенентами

Установка: `pip3 install objection`

Запуск: `objection -g "package_name" explore`



```
Windows PowerShell
PS C:\Users\kerby> objection -g "b3nac.injuredandroid" explore
Using USB device `Nokia 7 2`
Agent injected and responds ok!

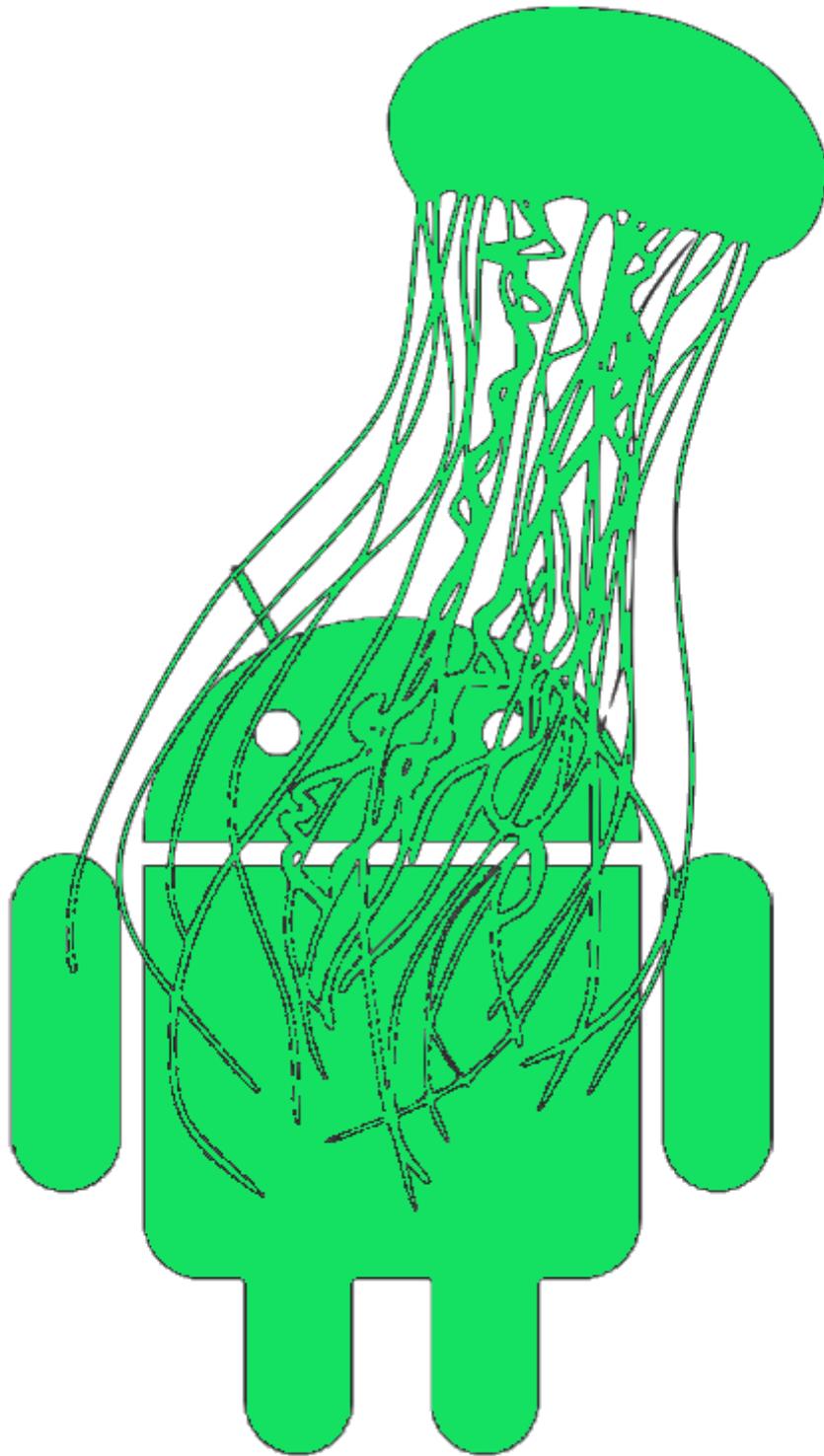
  _ _ _ _ _
 | . | . | . | . | . | . | . | . | . |
 | _ | _ | _ | _ | _ | _ | _ | _ | _ |
 | _ _ | (object)inject(ion) v1.11.0

Runtime Mobile Exploration
by: @leonjza from @sensepost

[tab] for command suggestions
b3nac.injuredandroid on (Nokia: 11) [usb] # android hooking list activities
b3nac.injuredandroid.AssemblyActivity
b3nac.injuredandroid.CSPBypassActivity
b3nac.injuredandroid.ContactActivity
b3nac.injuredandroid.DeepLinkActivity
b3nac.injuredandroid.DisplayPostXSS
b3nac.injuredandroid.ExportedProtectedIntent
b3nac.injuredandroid.FlagEightLoginActivity
b3nac.injuredandroid.FlagFiveActivity
b3nac.injuredandroid.FlagFourActivity
b3nac.injuredandroid.FlagNineFirebaseActivity
b3nac.injuredandroid.FlagOneLoginActivity
b3nac.injuredandroid.FlagOneSuccess
b3nac.injuredandroid.FlagSevenSqliteActivity
b3nac.injuredandroid.FlagSeventeenActivity
b3nac.injuredandroid.FlagSixLoginActivity
```

А еще у него просто шикарнейшие автоподсказки.

## Medusa



## MEDUSA

В моем представлении, это андерграунд среди средств для анализа мобильных приложений. Он мало где упоминается, что очень зря. Это самая большая коллекция скриптов для Frida из тех, что я видел. Можно сравнить с metasploit по подходу: у вас есть коллекция скриптов, вы выбираете необходимые, они компилируются вместе и инжектятся в процесс.

```

medusa> use
JNICalls/CallObjectMethod      encryption/cipher_2             helpers/get_system_properties
JNICalls/DefineClass           encryption/cipher_3             helpers/keystore_extract
JNICalls/FindClass             encryption/hash_operations      helpers/translator
JNICalls/GetByteArrayRegion    exploits/log4j                  helpers/unlinker
JNICalls/GetMethodID           file_system/asset_manager      http_communications/certificate_pinner_builder
JNICalls/NewObject             file_system/file_exists        http_communications/intercept_json_objects
JNICalls/NewStringUTF          file_system/file_write         http_communications/libssl_ssl_set_custom_verify
JNICalls/RegisterNatives       file_system/input_output       http_communications/multiple-unpinner_v2
JNICalls/ReleaseStringUTFChars file_system/prevent_delete     http_communications/multiple_unpinner
JNICalls/SetByteArrayRegion    file_system/shared_preferences http_communications/okhttp3_retrofit
JNICalls/SetCharArrayRegion    file_system/storage            http_communications/universal_SSL_pinning_bypass
JNICalls/SetShortArrayRegion  firebase/database_reference    http_communications/uri_logger
backdoor/backdoor_calls       firebase/firebase_authentication http_communications/volley_request
base64/base64_interceptor     firebase/firebase_firestore    ipc/incoming_intents
bluetooth/bluetooth          firebase/firebase_messaging    ipc/intent_creation_monitor
clickers/click_toll_fraud     flutter/verify_cert_chain_bypass_v7a ipc/ipc
clipboard/clipboard          flutter/verify_cert_chain_bypass_v8a ipc/outgoing_intents
code_loading/dump_dex         flutter/verify_cert_chain_bypass_x86_64 runtime/runtime
code_loading/dynamic_code_loading helpers/android_debug_log      scratchpad
code_loading/native_libs     helpers/anti_debug             services/accessibility_nod
compression/gzip_input_stream helpers/cancel_system_exit     services/notification_listener
cordova/cordova_enable_debugging helpers/de_reflector           sms_fraud/sms_fraud
cordova/get_loaded_plugins    helpers/de_reflector_2        sockets/socket_monitor
db_queries/SQLiteDatabase    helpers/device_cloaking       sockets/socket_monitor_2
db_queries/content_provider_query helpers/enable_buttons         spyware/keylogger
db_queries/db                helpers/enable_screenshot     spyware/spyware_hooks
encryption/cipher_1          helpers/enumerate_loaded_classes webviews/hook_webviews

```

Очень удобно и иногда экономит кучу времени. Однако пару раз в модулях находились опечатки и ошибки, из-за чего скрипты ломались, так что приходилось править руками :)

## Magisk-модули

Если посмотреть на Magisk-модули, то не так уж и много необходимых нам, которые позволяют сэкономить время.

### MagiskTrustUserCerts

<https://github.com/NVISOsecurity/MagiskTrustUserCerts>

Таким модулем является MagiskTrustUserCerts. Современные версии Android не дают возможности добавить системный CA. Это надо делать вручную или при помощи таких модулей, которые на этапе загрузки системы перенесут клиентские сертификаты в системный каталог. Это необходимо, чтобы приложения доверяли нам, когда мы заворачиваем трафик на прокси для последующего анализа.

Данный модуль уже упоминался нами, когда мы настраивали WSA.

## XPosed-модули

Ввиду устаревания оригинального XPosed, мы используем его форк — LSpoted. Данный форк — это модуль для Magisk, из-за чего установка производится буквально в несколько кликов.

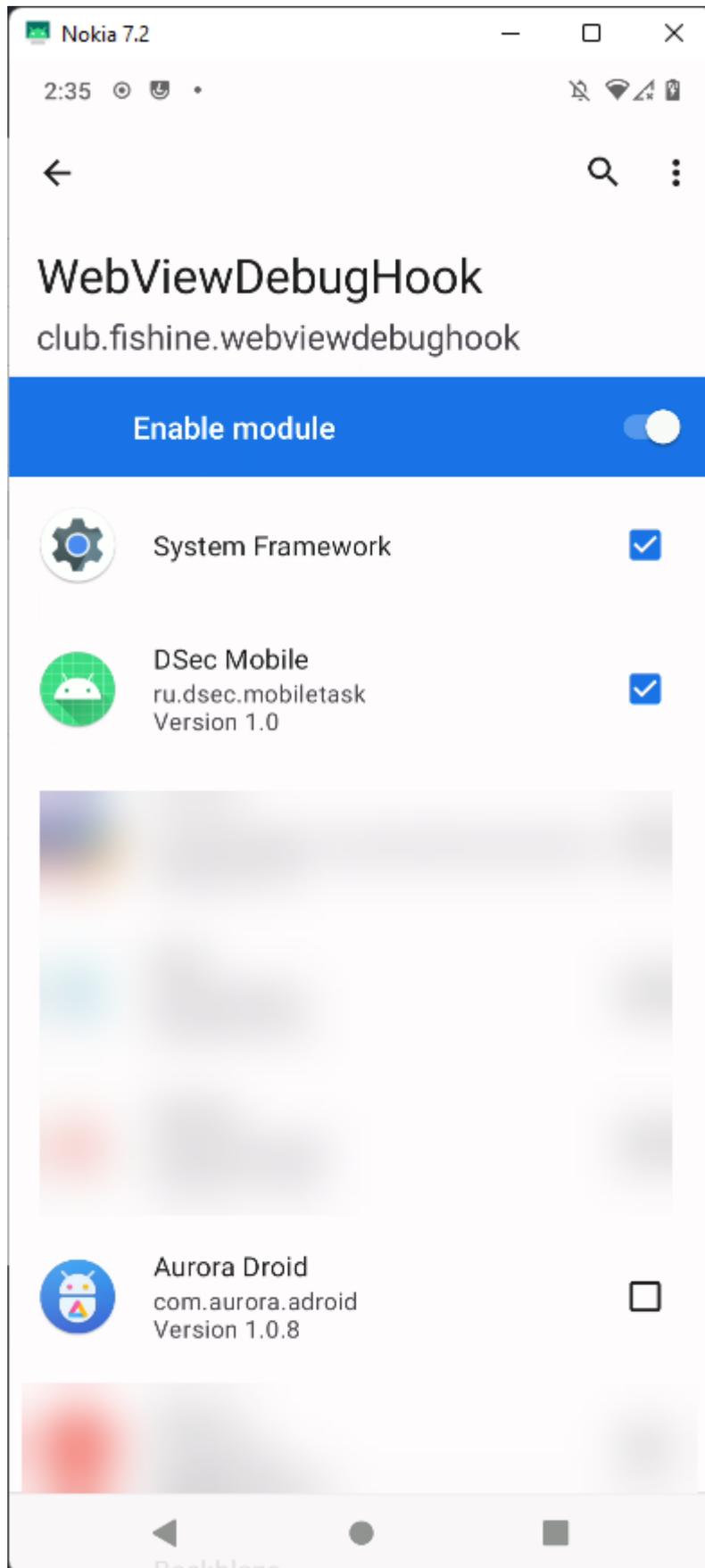
### WebViewDebugHook

<https://github.com/feix760/WebViewDebugHook>

Мало что можно написать про этот модуль, но он делает буквально следующее:

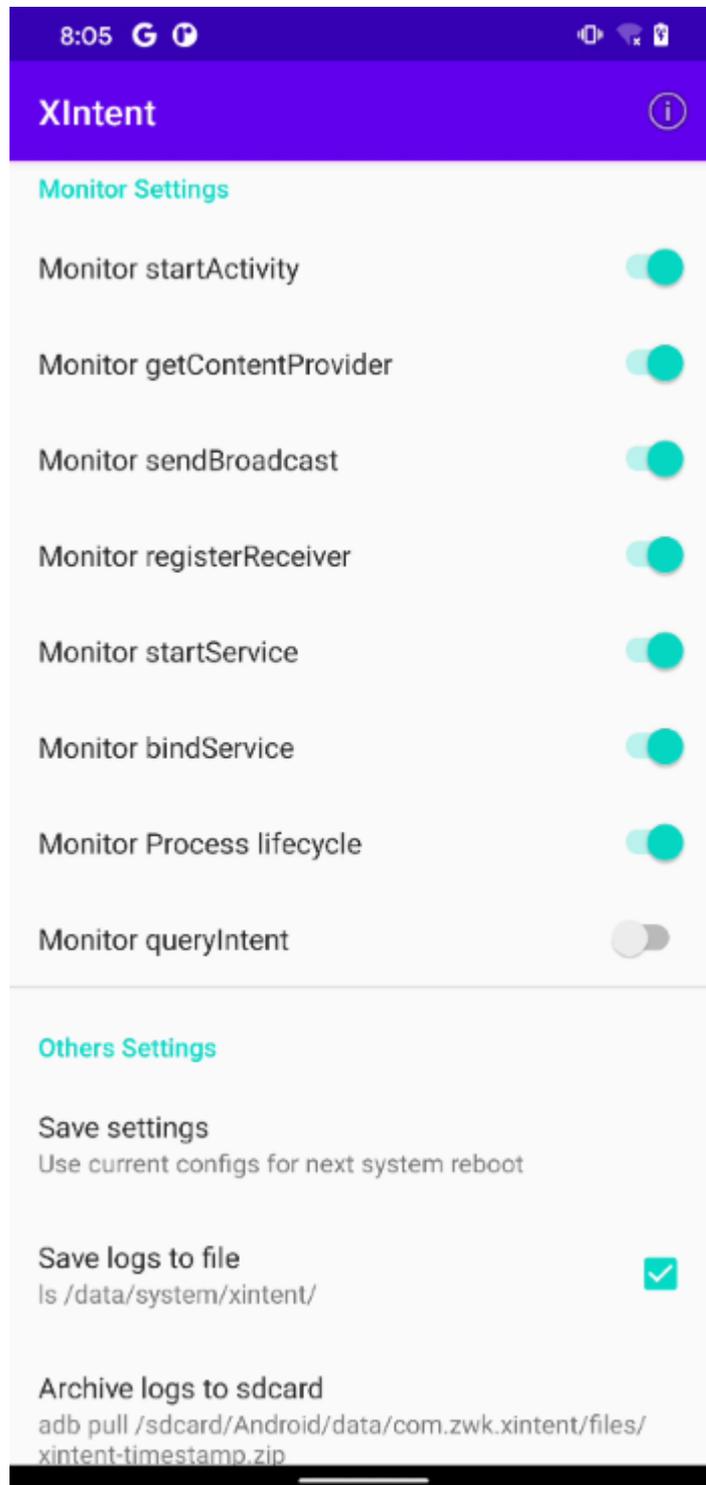
```
WebView.setWebContentsDebuggingEnabled(true);
```

Удобно, если приложение использует WebView и нужно взглянуть на нее "изнутри". Устанавливаем данный модуль, через LSposed выбираем скоуп приложений, на которые будет распространяться данный модуль.



Теперь, когда у нас на активном экране будет WebView, мы сможем посмотреть на нее через Chrome ( `chrome://inspect` ).





Как и раньше, нужно выбрать целевое приложение — и вперед смотреть логи.

```
XIntentLogger: startActivity { calling=com.zwk.xintenttest:10222--1, rc=10222-10999, ITS=false, requestCode=0, startFlags=0, target=com.zwk.xintenttest/com.zwk.xintenttest.priv.URIReActivity<false>
>, intent=Intent { act=genTestIntent1 cmp=com.zwk.xintenttest/priv.URIReActivity (has extras) }, extras={ barray:([D]2:[true, false]), iarray:([D]3:[1, 2, 3]), ivalue:(java.lang.Integer)999, sarray:([Ljava.lang.String;)[A, B], subIntent:(android.content.Intent)subIntent#1: { Intent { act=THIS.IS.A.SUBINTEENT } }, barray2:([([Z]2:[true], [false, true]), larray2:([([J]2:[1999999999], [599999999, 3999999]), bv:(java.lang.Byte)1, barr:([B]4:[10, 11, 12, 13]), carr:([C]3:[x, y, z]), barr2:([([B]2:[10, 11], [12, 13]), empty:null, darray2:([([D]2:[13.14], [2.718, 0.618]), iarray2:([([I]2:[1], [2, 3]), sarray2:([([Ljava.lang.String;)[A, B], [C]), Bundle:(android.os.Bundle)subBundle#0 { bundle:(android.os.Bundle)subBundle#1 { sub:(java.lang.String)sub }, brr:([B]3:[97, 98, 99]), int:(java.lang.Integer)1, str:(java.lang.String)str, byte:(java.lang.Byte)7, long:(java.lang.Long)88888, IList:(java.util.ArrayList)[1, 2, 3], boaar:([Z]4:[false, false, true, true]) } } }
```

## ADB

Наверное, странно видеть тут ADB, но не упомянуть его нельзя. Он умеет многое и

играет роль чуть ли не основного инструмента при взаимодействии с устройством.

## Проброс портов

```
# пробрасываем 7777 порт на устройстве, чтобы трафик шел на 8080 порт хоста
adb reverse tcp:7777 tcp:8080
```

## Манипуляции с настройками

Командами ниже можно указать HTTP прокси и убрать его соответственно.

```
adb shell settings put global http_proxy 127.0.0.1:7777
adb shell settings put global http_proxy :0
```

## Текущая on-top activity и fragment

```
adb shell "dumpsys activity activities | grep mResumedActivity"
```

## Вывести список всех приложений и путь до base APK

```
adb shell pm list packages -f
```

## ADB logcat по package name

```
linux:
adb logcat --pid=`adb shell pidof -s com.example.app`

windows:
adb logcat --pid=$(adb shell pidof -s ru.dsec.mobiletask)
```

## Запустить intent

Базовый пример:

```
adb shell am start -a android.intent.action.VIEW -d "dsec://open"
```

am умеет практически все. У него длинный, но крайне понятный manual :)

## Прочее

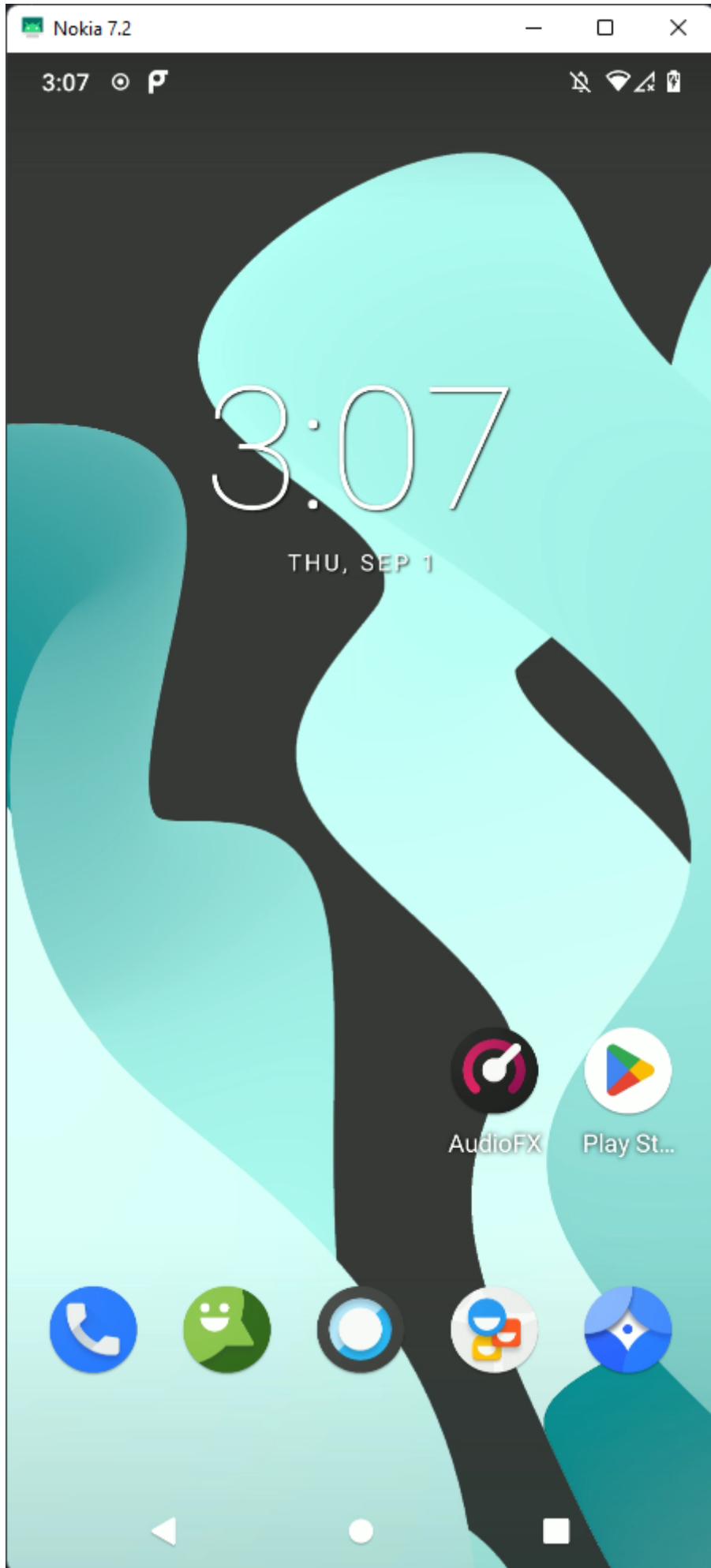
Сюда попало все, что не вошло в остальные категории.

## Scrcpy

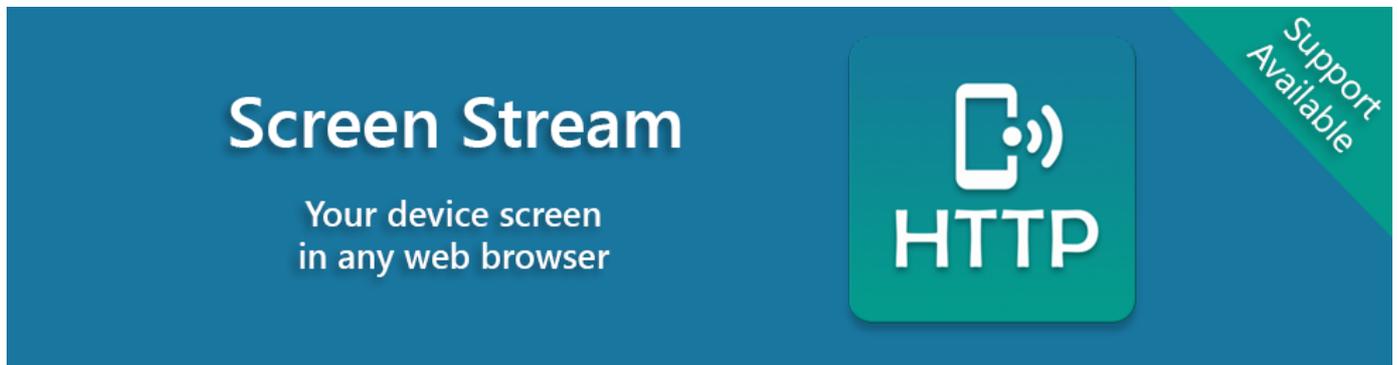
<https://github.com/Genymobile/scrcpy>

Это приложение позволяет отобразить экран Android-девайса, доступного через ADB. Также позволяет вводить данные через клавиатуру. Очень удобно, когда нужно вводить огромный пароль от тестовой УЗ и делать это через телефон крайне лениво :)

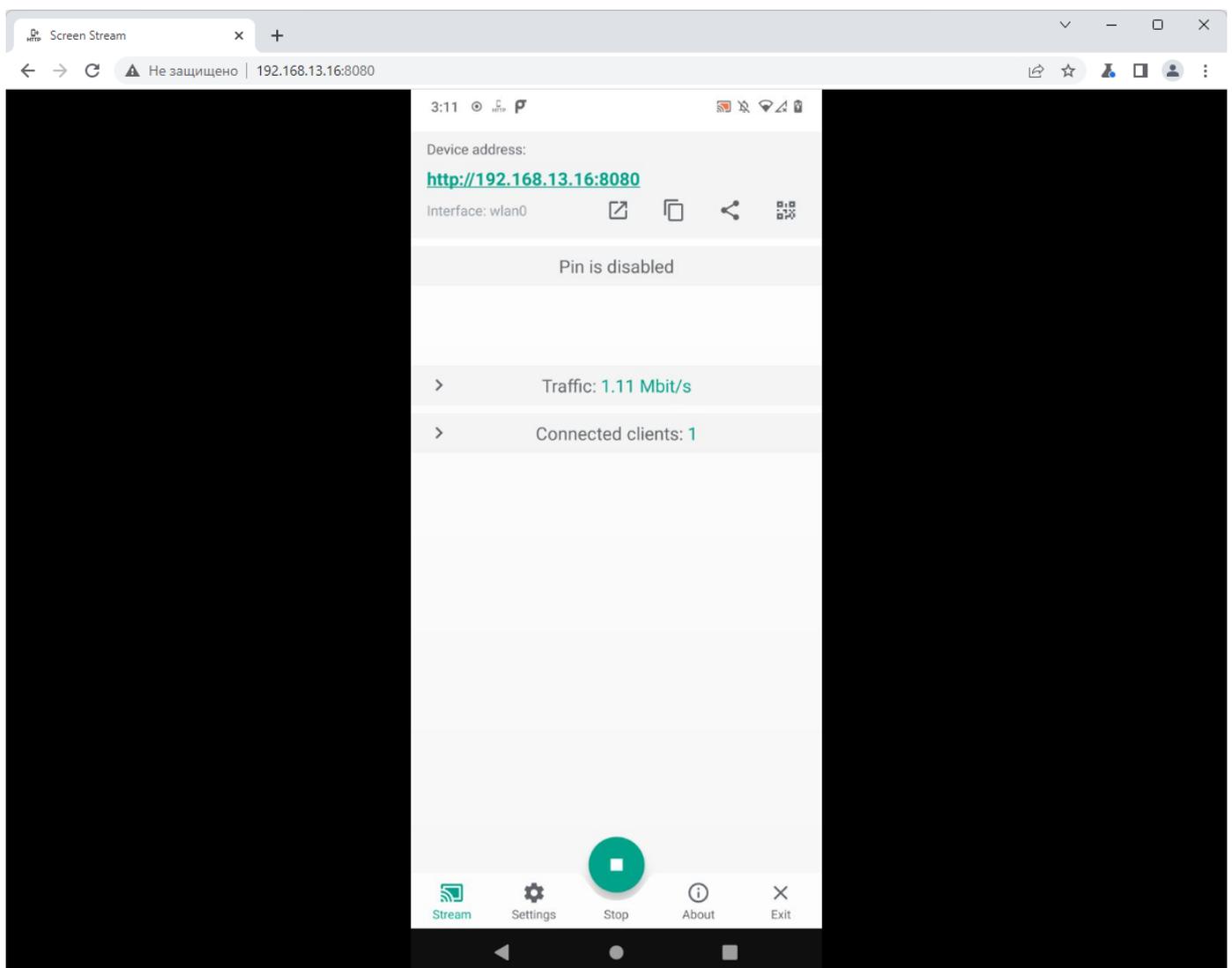
Есть под все ОС, что довольно редко для наших дней.



## ScreenStream



Казалось бы, зачем он нужен, когда есть scrcpy. Но что если нет возможности подключиться по проводу и вы находитесь в одной сети? Данное приложение нужно не очень часто, но помнить о нем полезно.



Работает с полпинка, главное — иметь сетевой доступ.

## Заключение

В этой статье представлены не все, но большинство инструментов, которые мы используем каждый день при работе с мобильными приложениями. Надеемся, что это поможет людям, которые только погружаются в анализ мобильных приложений и хотят собрать первое рабочее окружение.

Есть что дополнить? Пишите в комментариях.

**Теги:** [android](#), [pentest](#), [инфобез](#), [аудит безопасности](#), [статический анализ](#), [динамический анализ](#), [мобильные приложения](#)

**Хабы:** [Блог компании Digital Security](#), [Информационная безопасность](#), [Разработка под Android](#), [Тестирование мобильных приложений](#)

## Редакторский дайджест



Присылаем лучшие статьи раз в месяц



**Digital Security**

Безопасность как искусство

[Сайт](#) [Facebook](#) [Twitter](#) [Github](#) [Telegram](#) [ВКонтакте](#)



17

27

Карма    Рейтинг

**kerby @kerbyj**

Пользователь

## Комментарии 3



**drhouse**

13.09.2022 в 08:23

Однозначно в закладки! Спасибо!



+2

[ОТВЕТИТЬ](#)



 **whoam1ns3**  
22.09.2022 в 23:33

Там оно и останется навсегда

 0 [Ответить](#)



 **whoam1ns3**  
22.09.2022 в 23:33

А почему нет genumotion? Меньшее потребление ресурсов, рут из коробки...

 0 [Ответить](#)



Только полноправные пользователи могут оставлять комментарии. [Войдите](#), пожалуйста.

#### ПОХОЖИЕ ПУБЛИКАЦИИ

29 июля в 16:54

#### Анализ iOS-приложений

 +7

 2.4K

 26

 2 +2

19 ноября 2018 в 05:00

#### Клонируем бесконтактную карту с помощью мобильного приложения

 +42

 131K

 256

 21 +21

2 августа 2017 в 11:04

#### Внедряем безопасность в процесс разработки крупного проекта

 +40

 21K

 145

 6 +6

#### ЛУЧШИЕ ПУБЛИКАЦИИ ЗА СУТКИ

сегодня в 01:20

#### Ошибки выбора MongoDB в качестве основной БД в стартапе

 +66

 13K

 55

 55 +55

сегодня в 13:23

**Кейс: Как ошибка в рекламе принесла прибыль в 600 000 рублей** +49 6.4K 7 14 +14

вчера в 20:19

**Когда понты дороже денег: Оживляем Java подделку iPhone 4s, и смотрим на что она способна** +34 11K 18 20 +20

вчера в 18:37

**Брюнетки против блондинок или как на мониторе показать цвет свечения светильника?** +33 3K 22 9 +9

сегодня в 02:20

**Ниже некуда? Продажи ПК и ноутбуков падают гораздо быстрее прогнозных значений** +28 10K 12 32 +32**Ваш аккаунт**

Войти

Регистрация

**Разделы**

Публикации

Новости

Хабы

Компании

Авторы

Песочница

**Информация**

Устройство сайта

Для авторов

Для компаний

Документы

Соглашение

Конфиденциальность

**Услуги**

Корпоративный блог

Медийная реклама

Нативные проекты

Образовательные программы

Стартапам

Мегапроекты

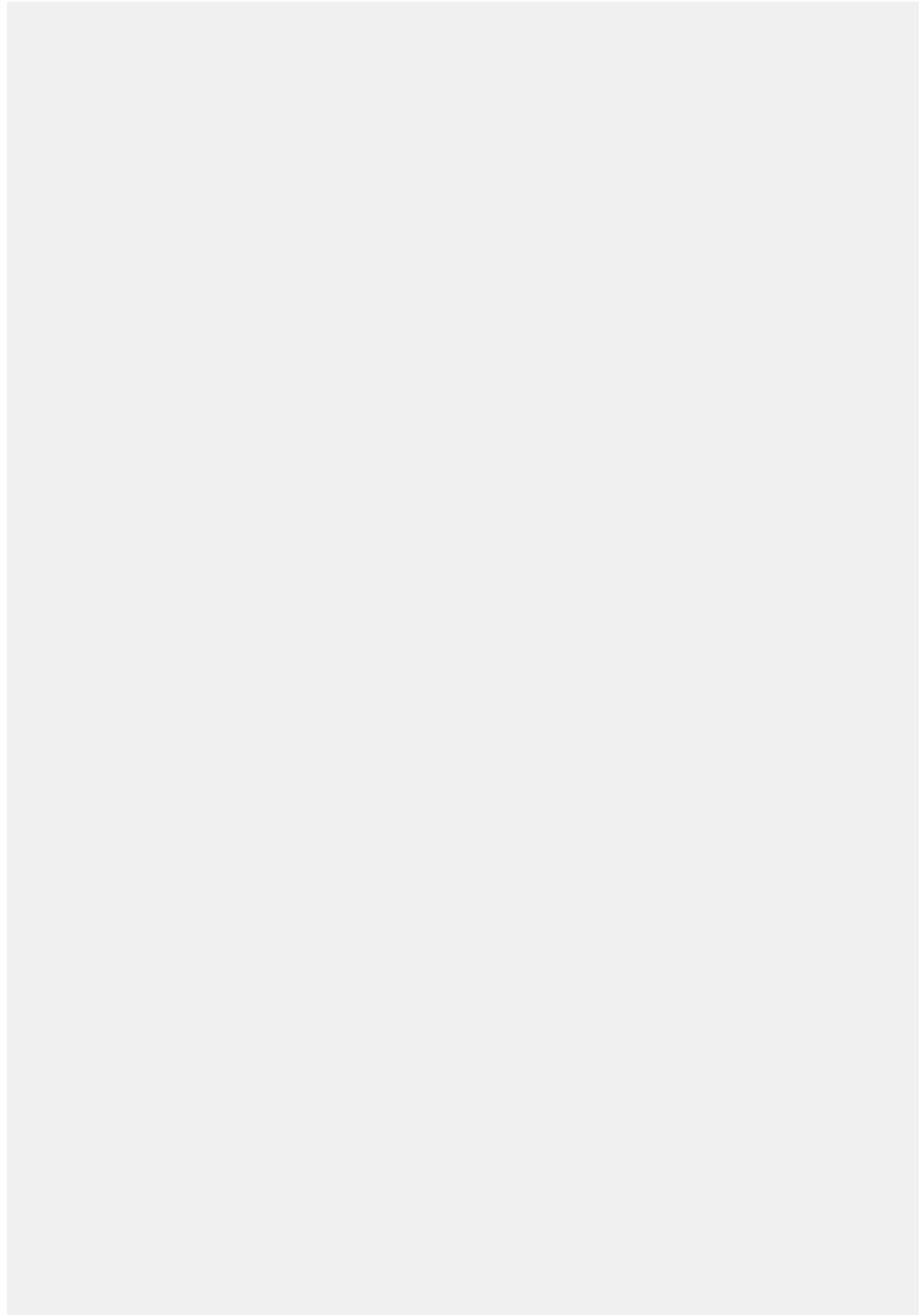


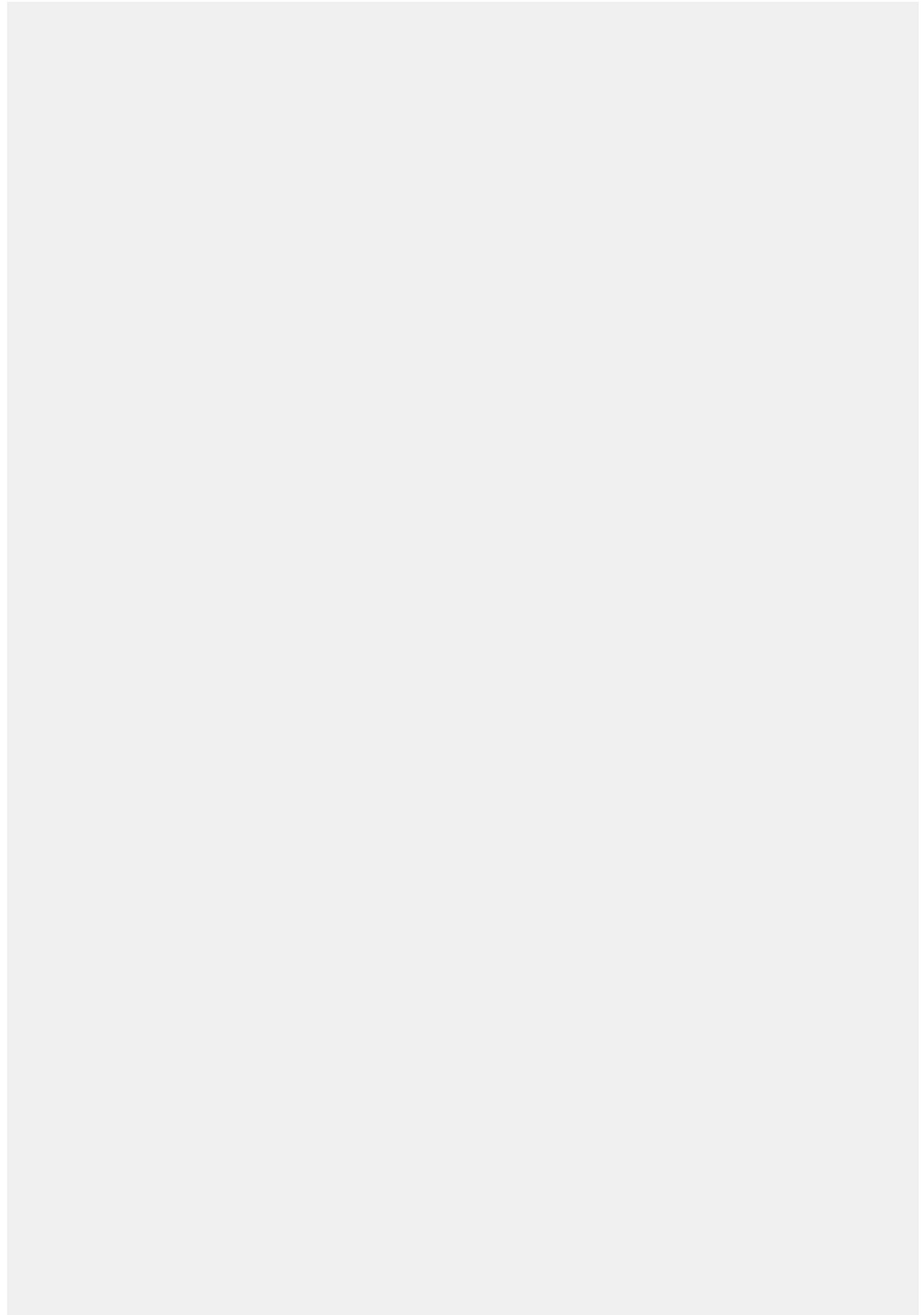
Настройка языка

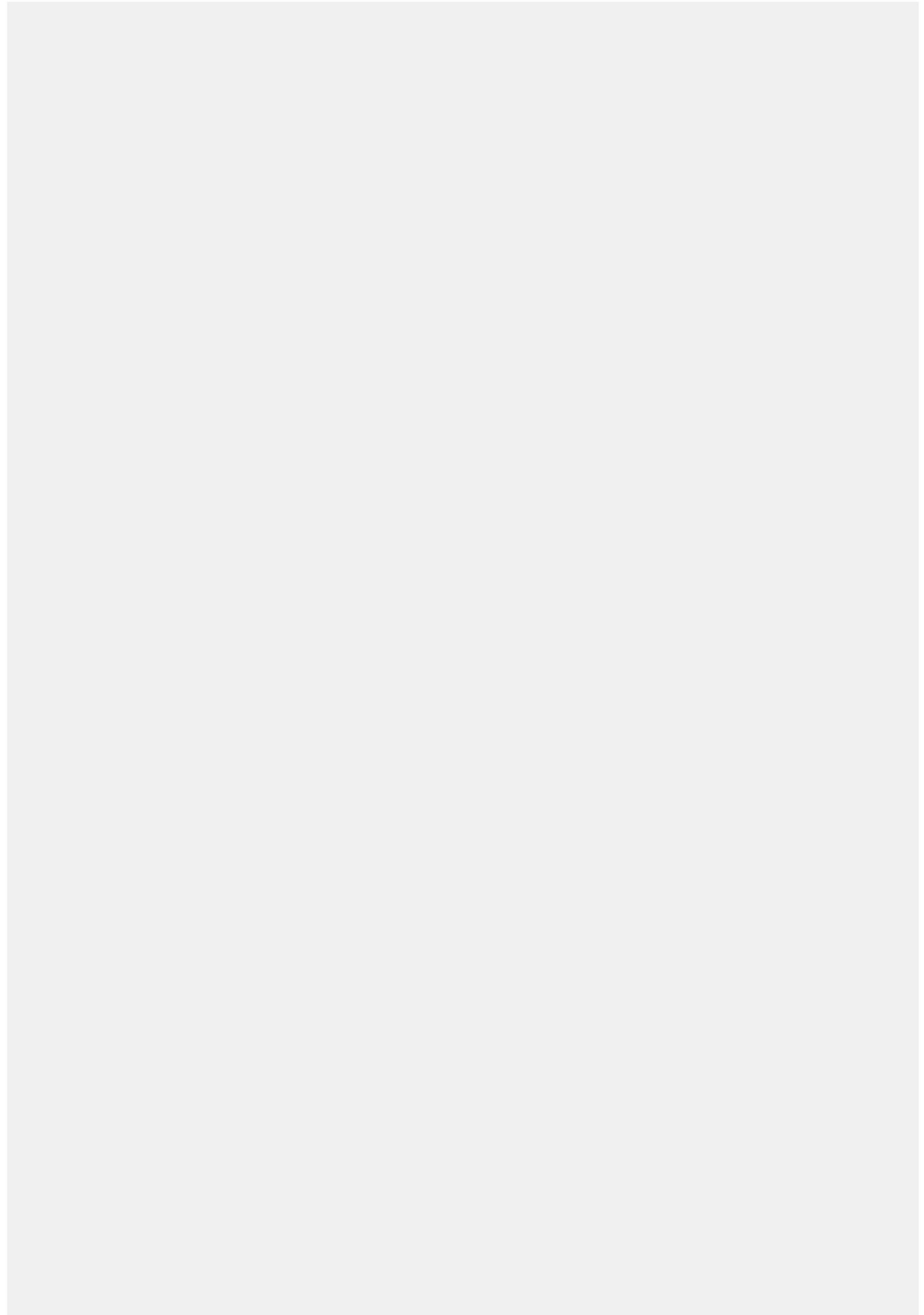
Техническая поддержка

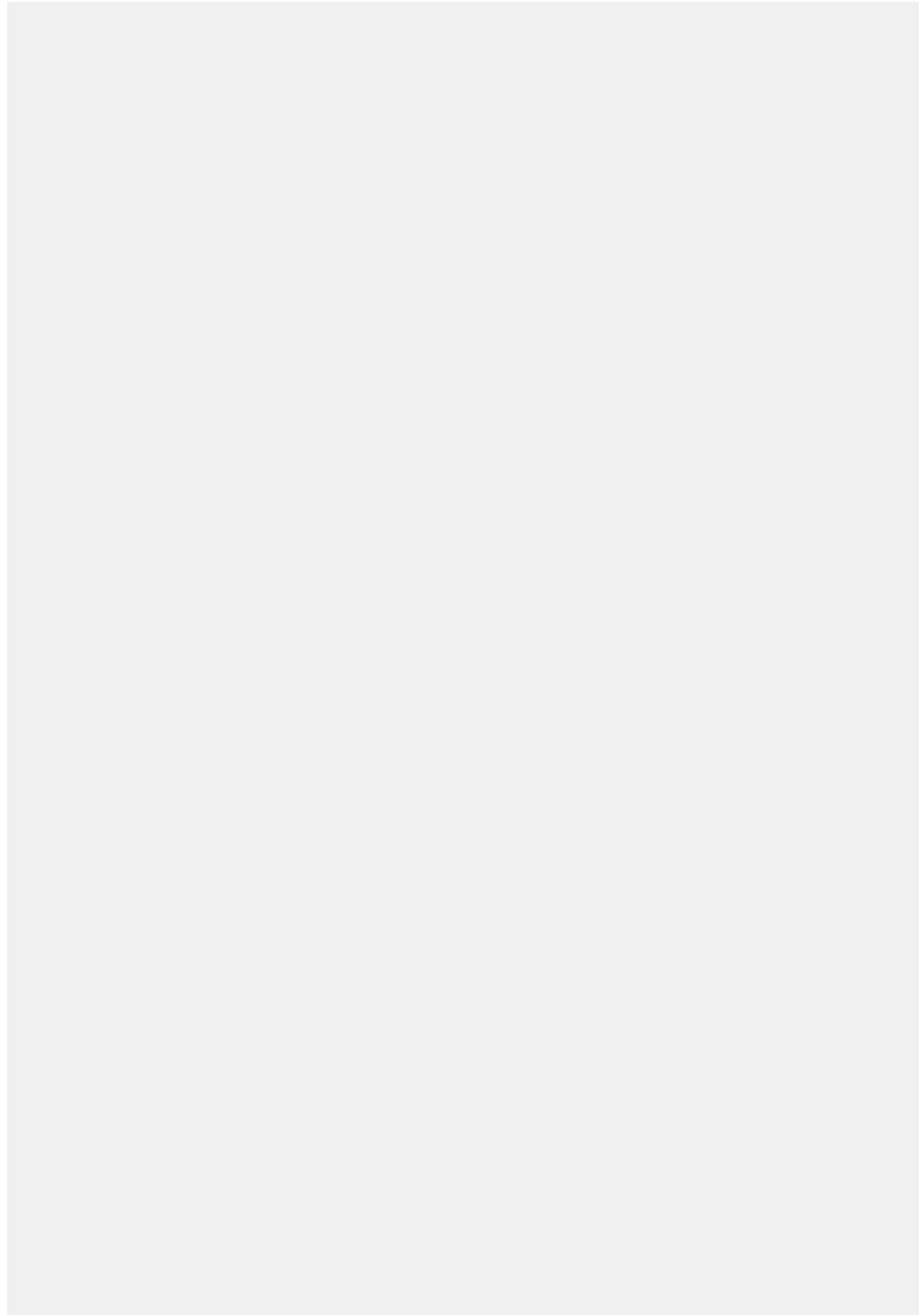
[Вернуться на старую версию](#)

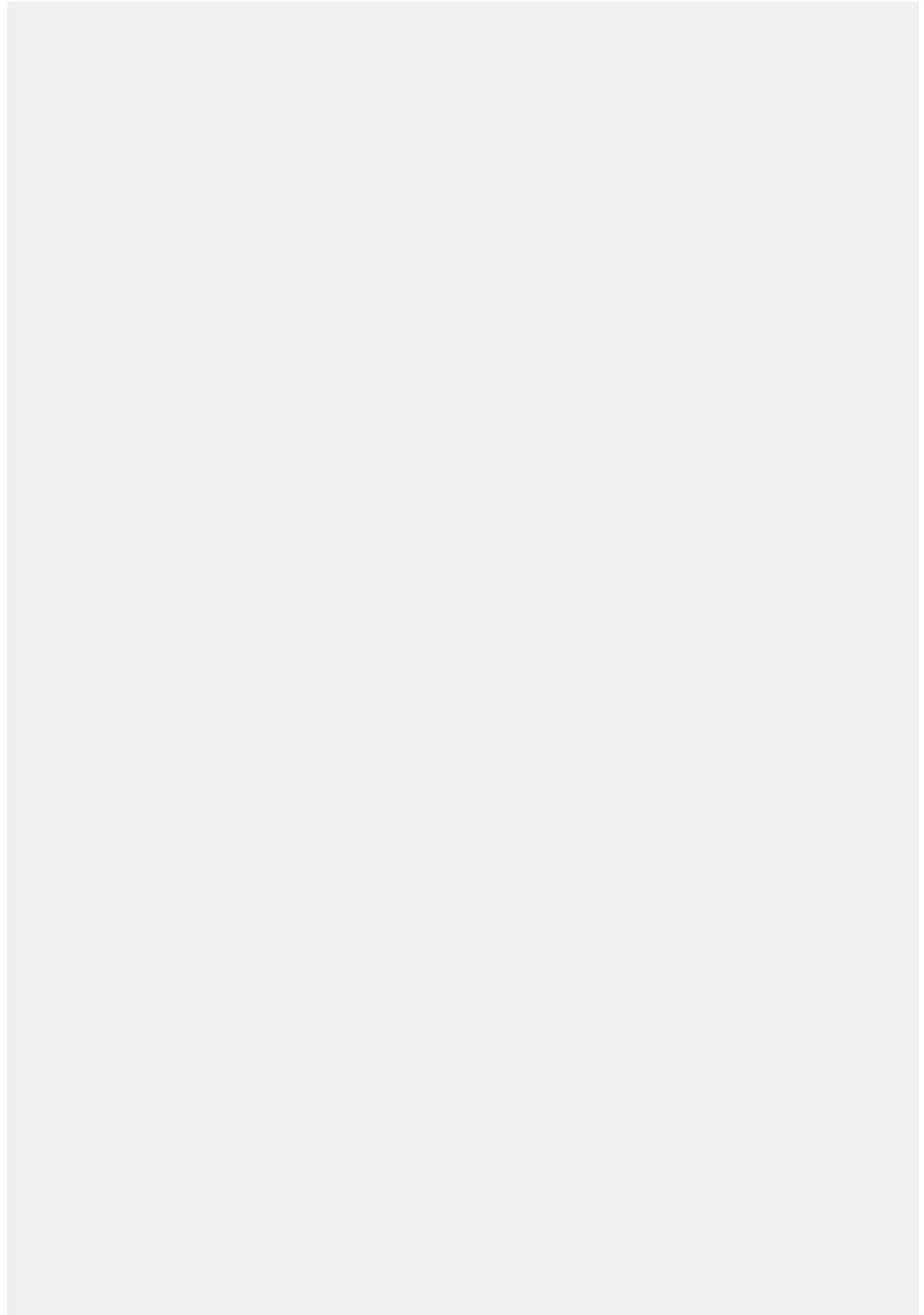
© 2006–2022, Habr

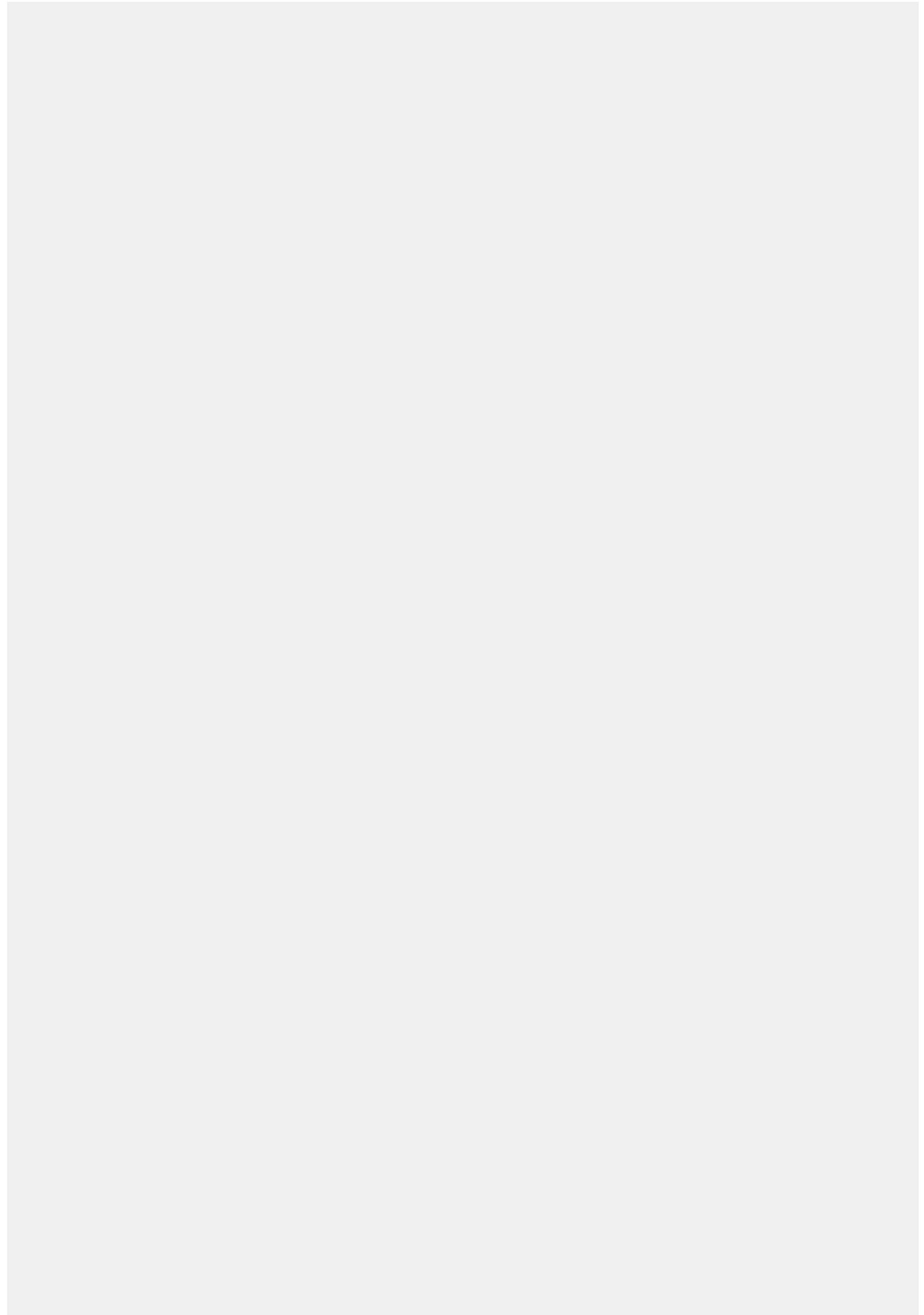


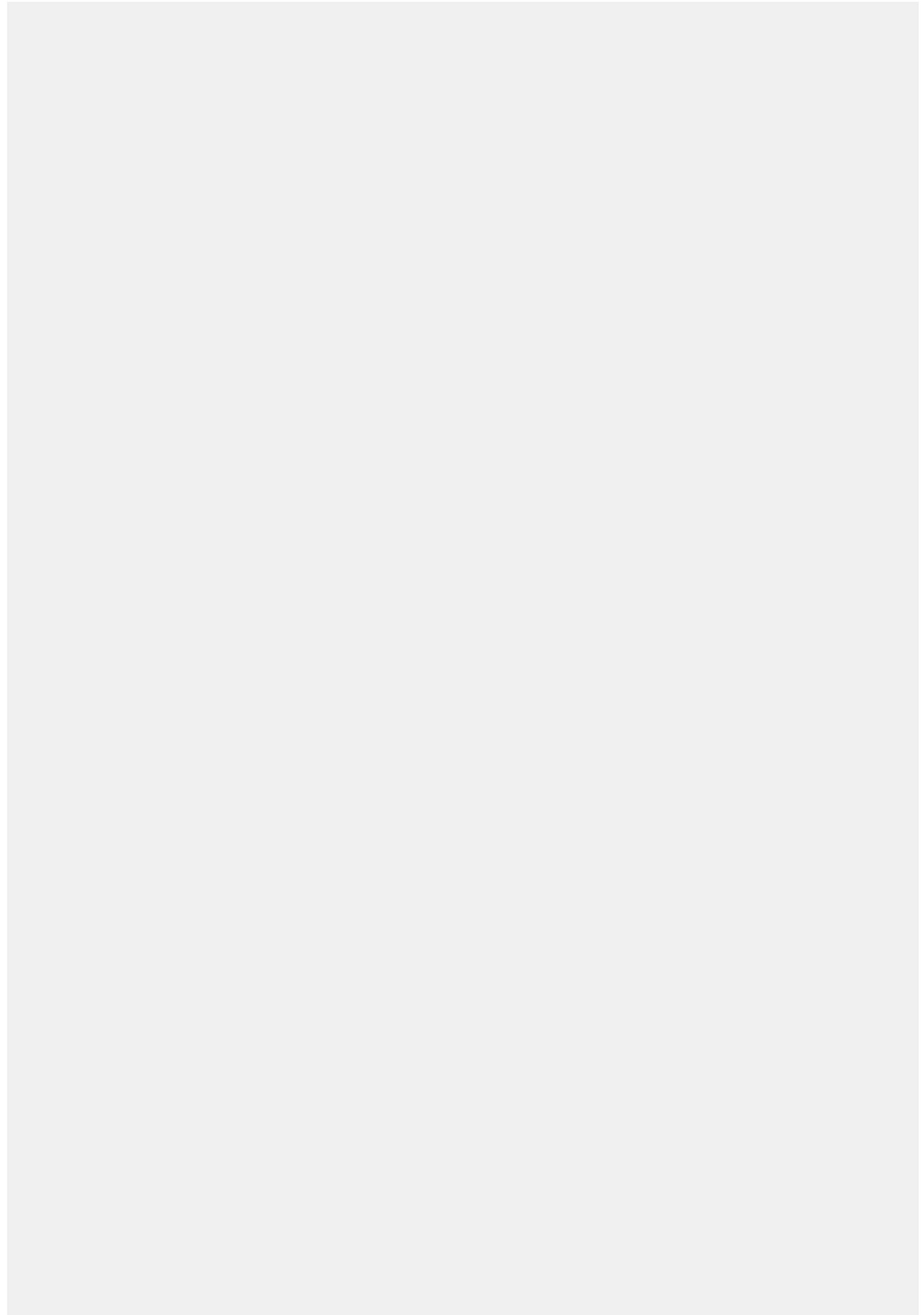


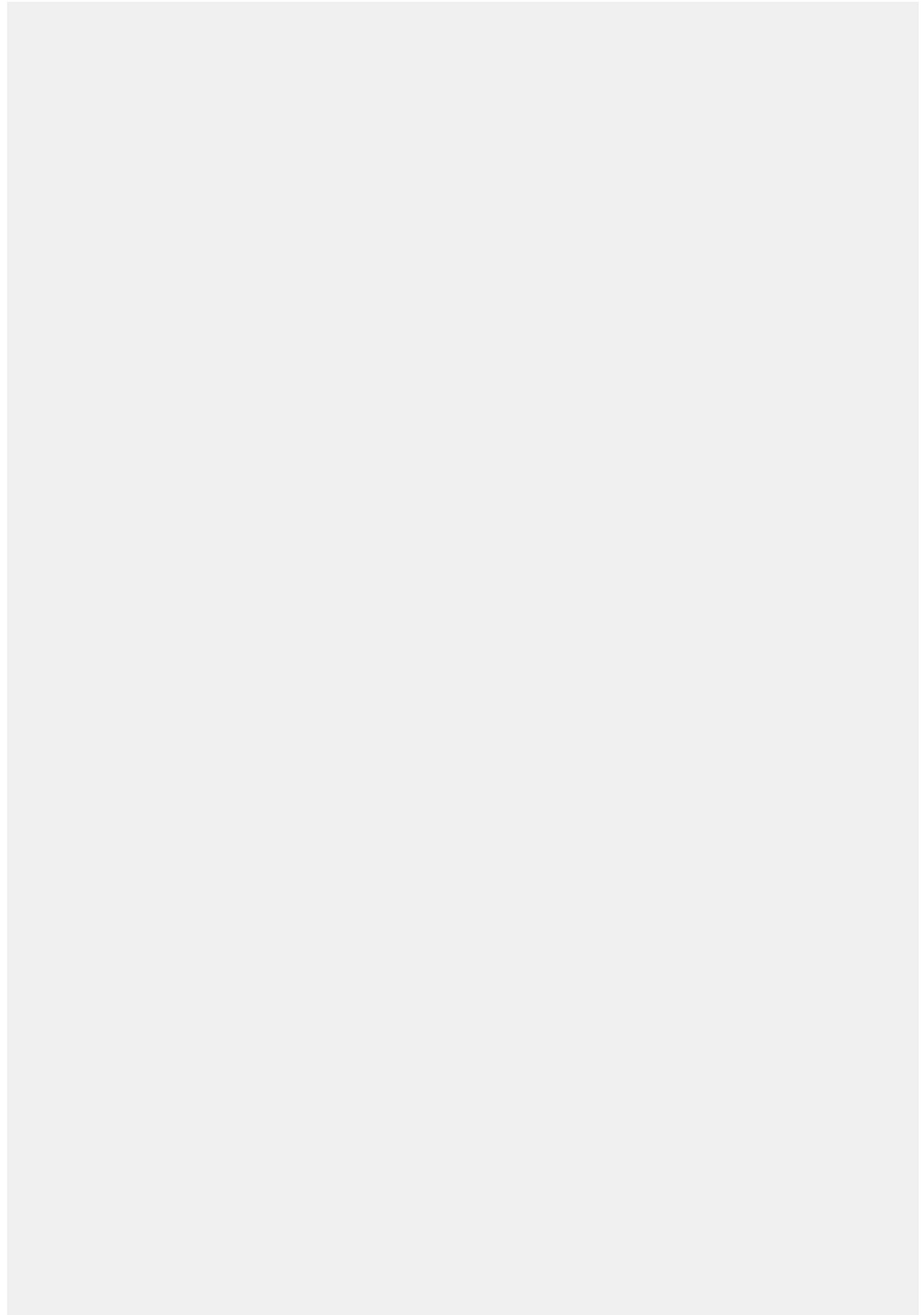


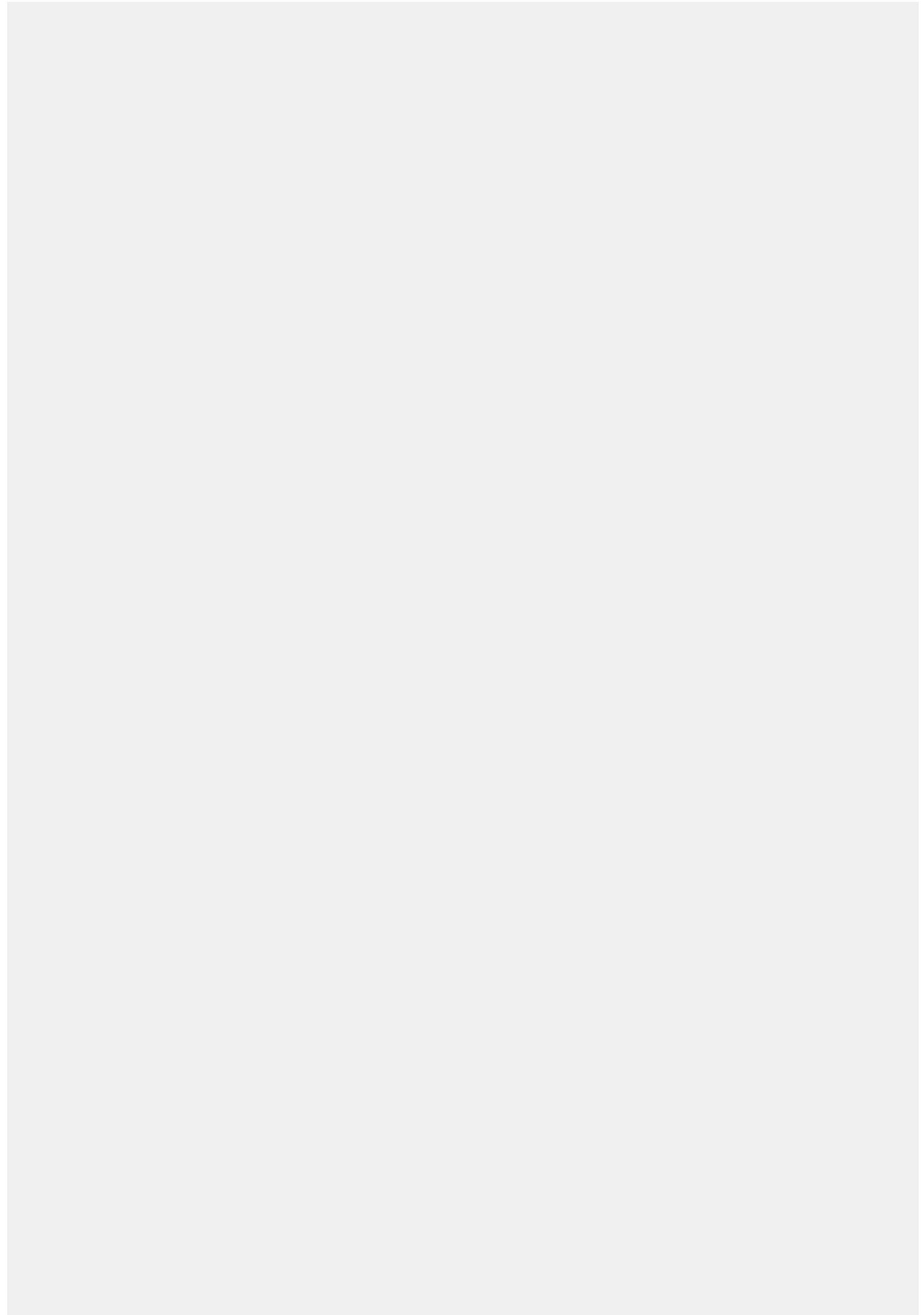


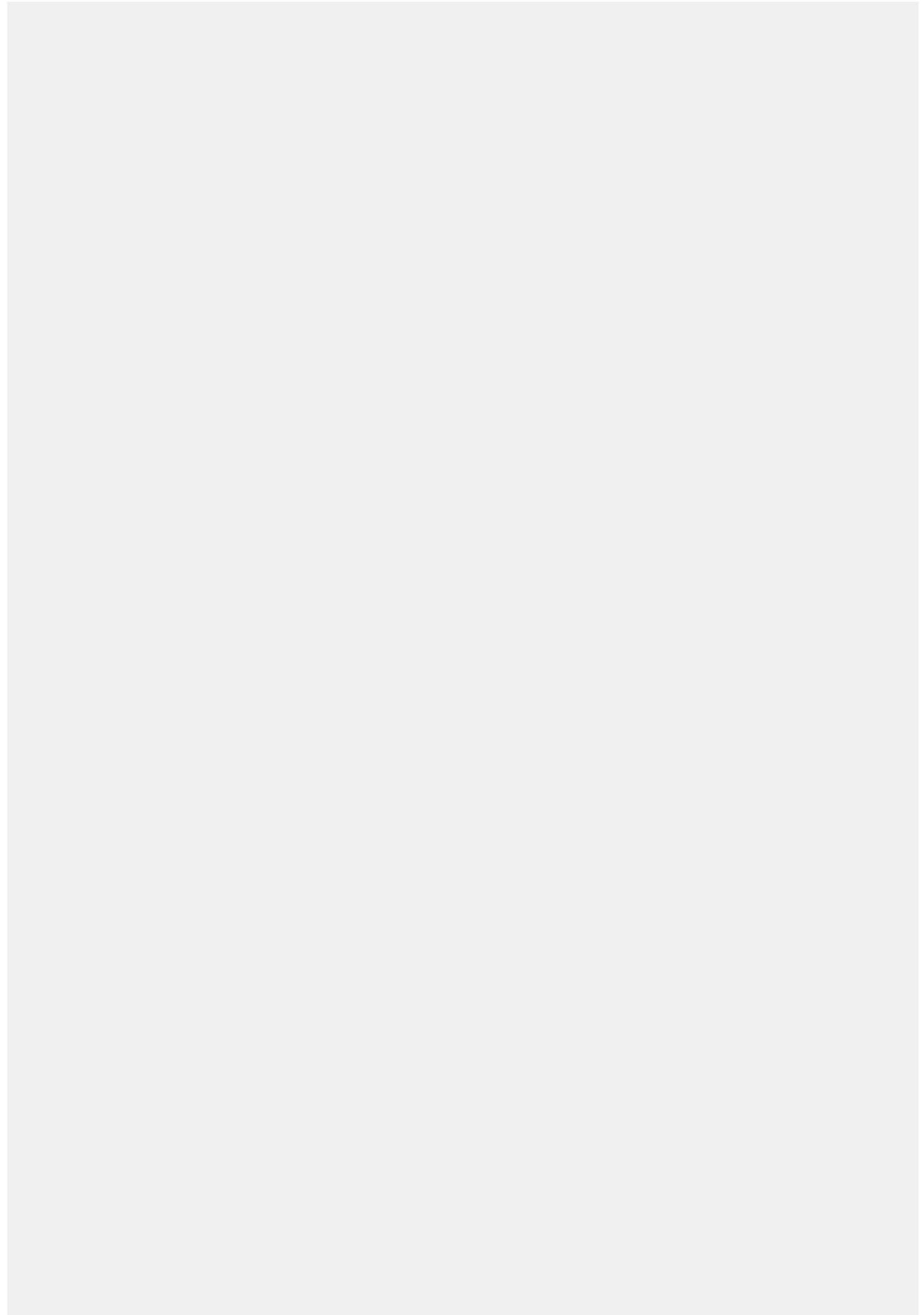


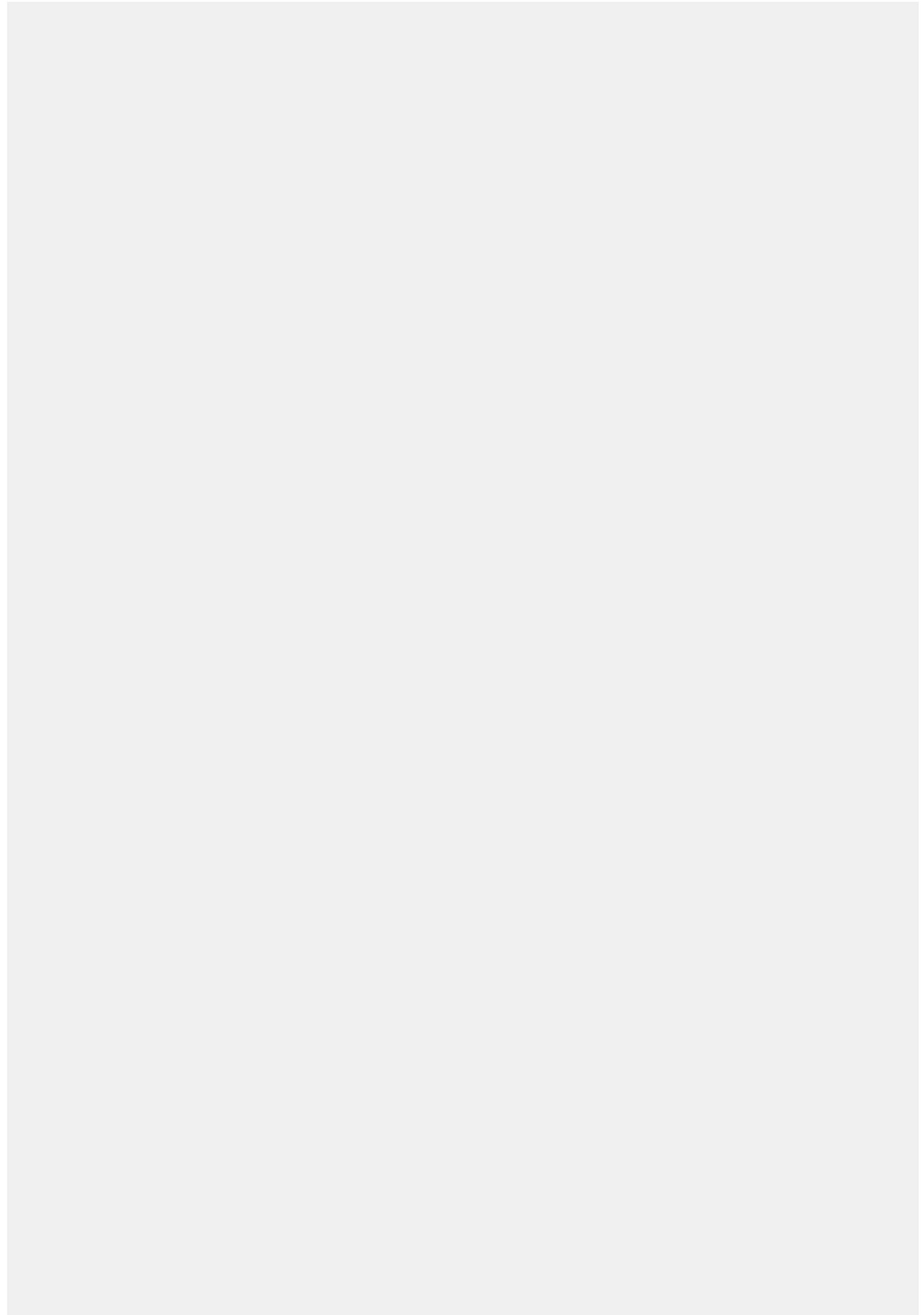


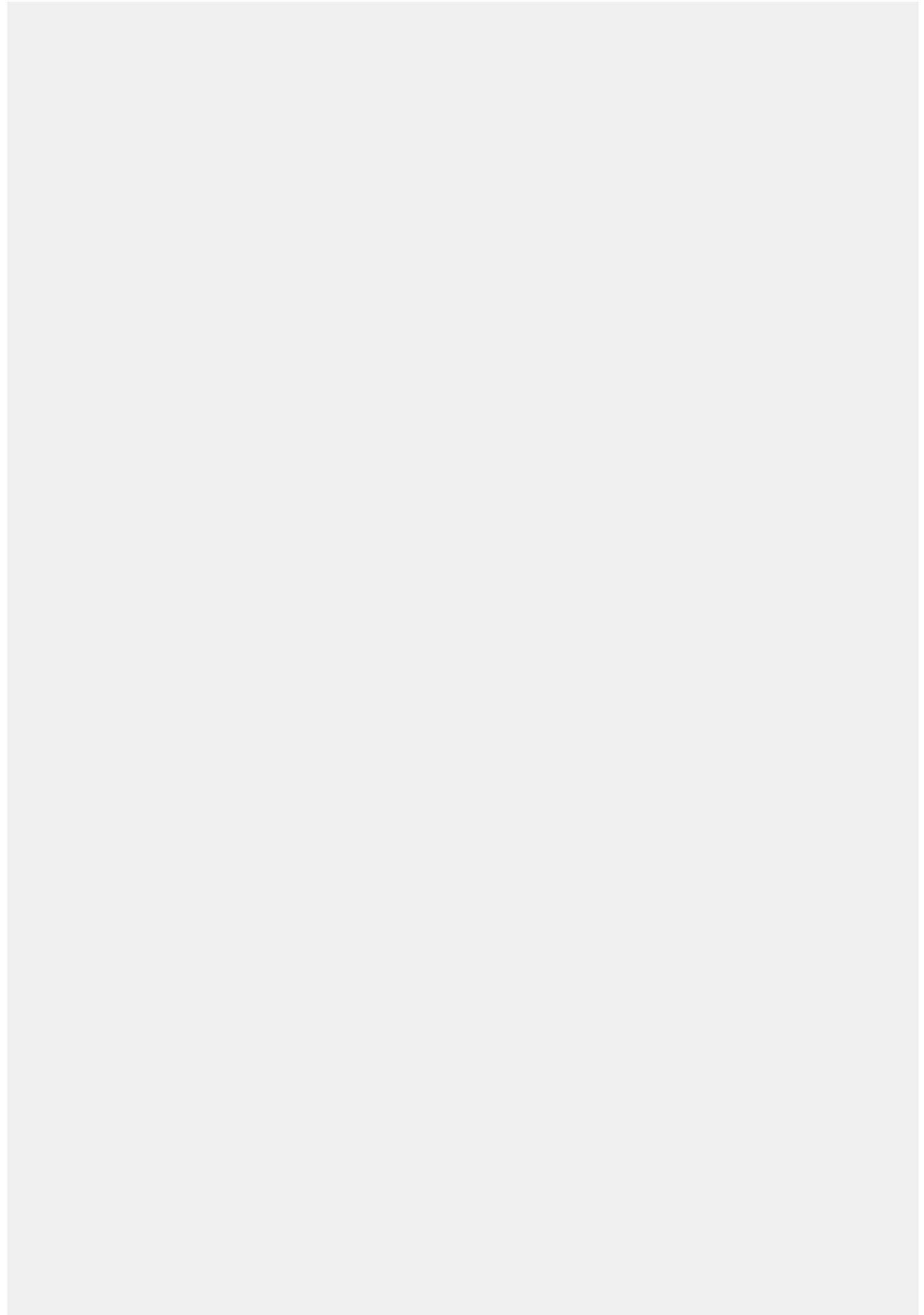


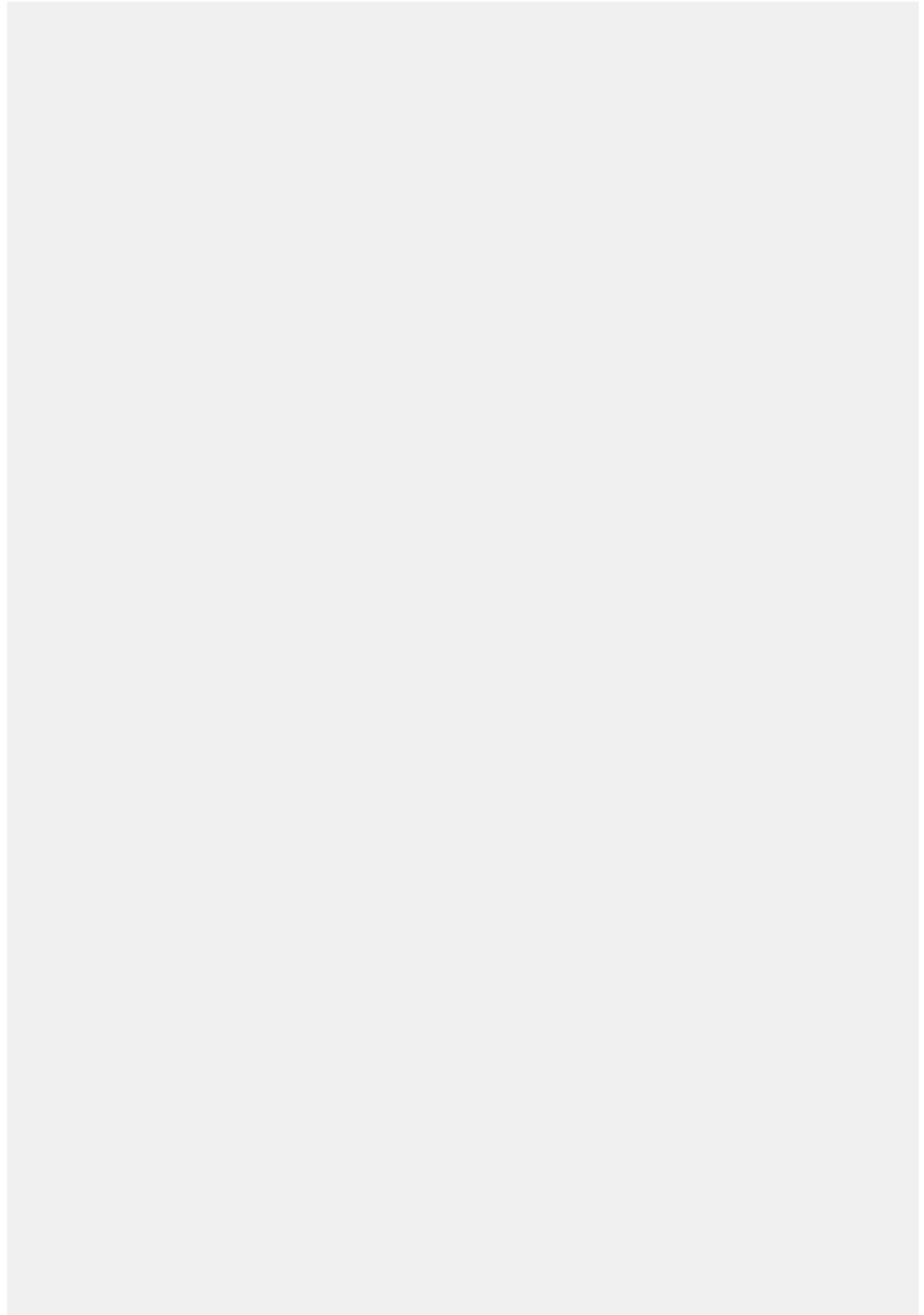


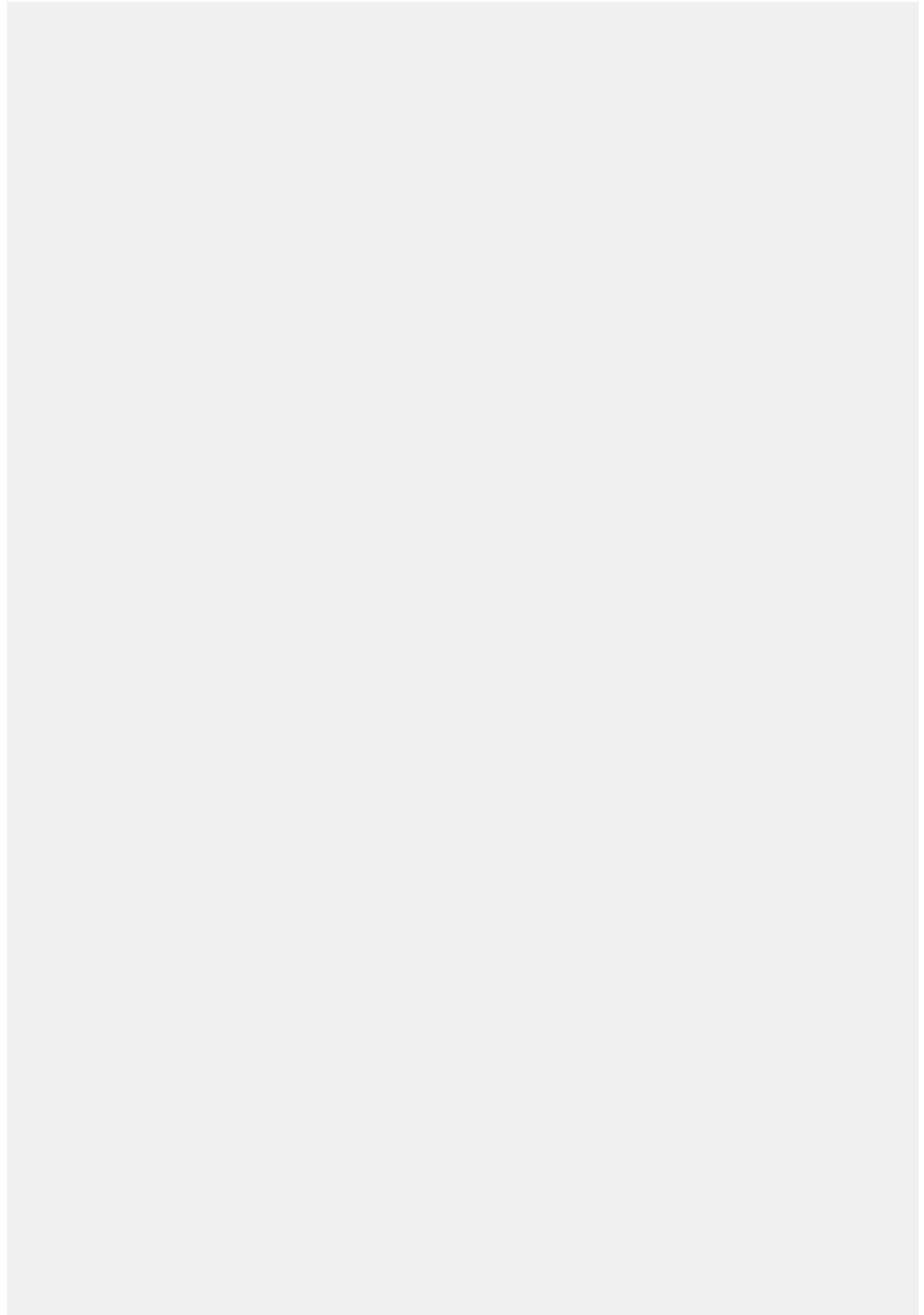


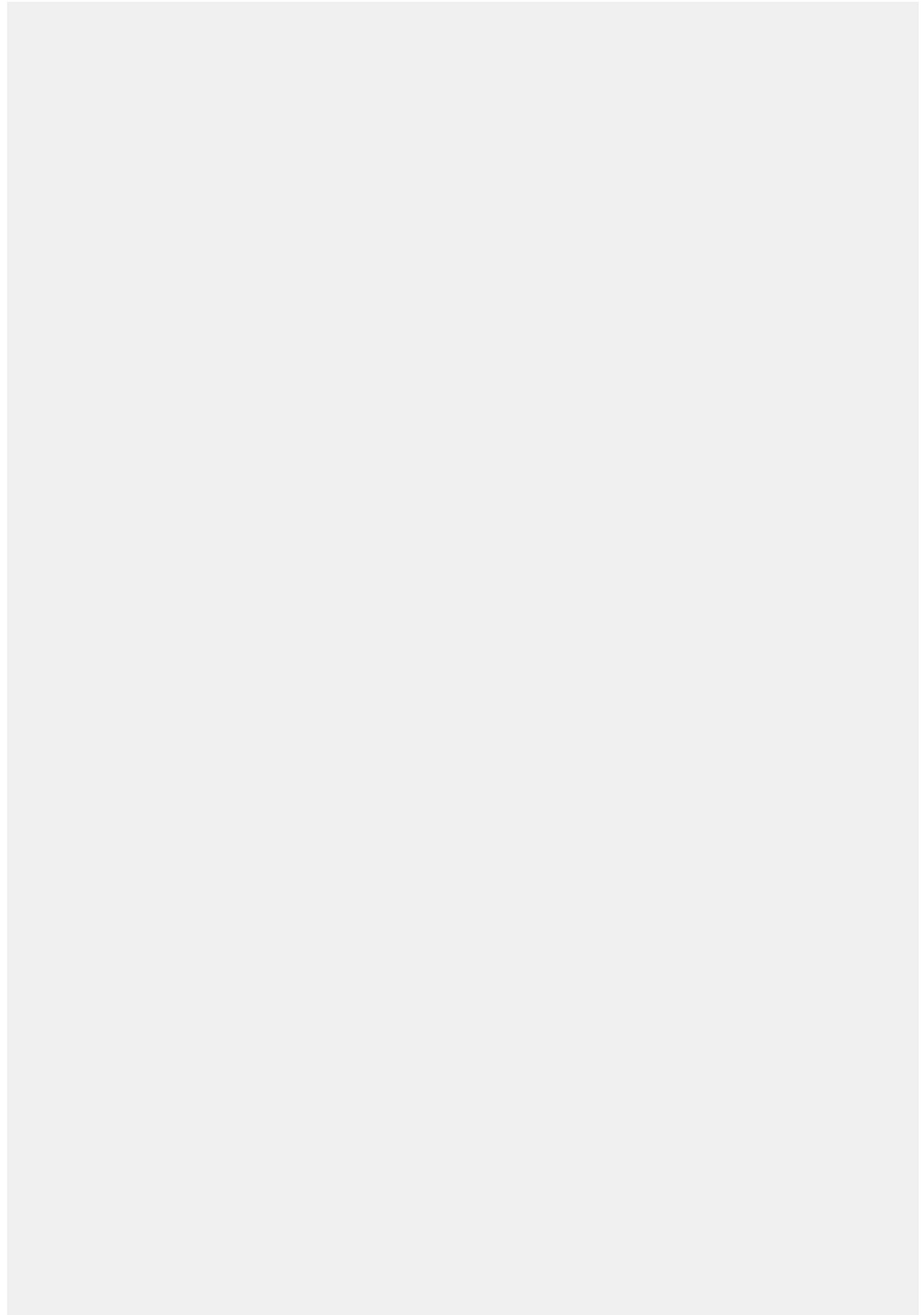


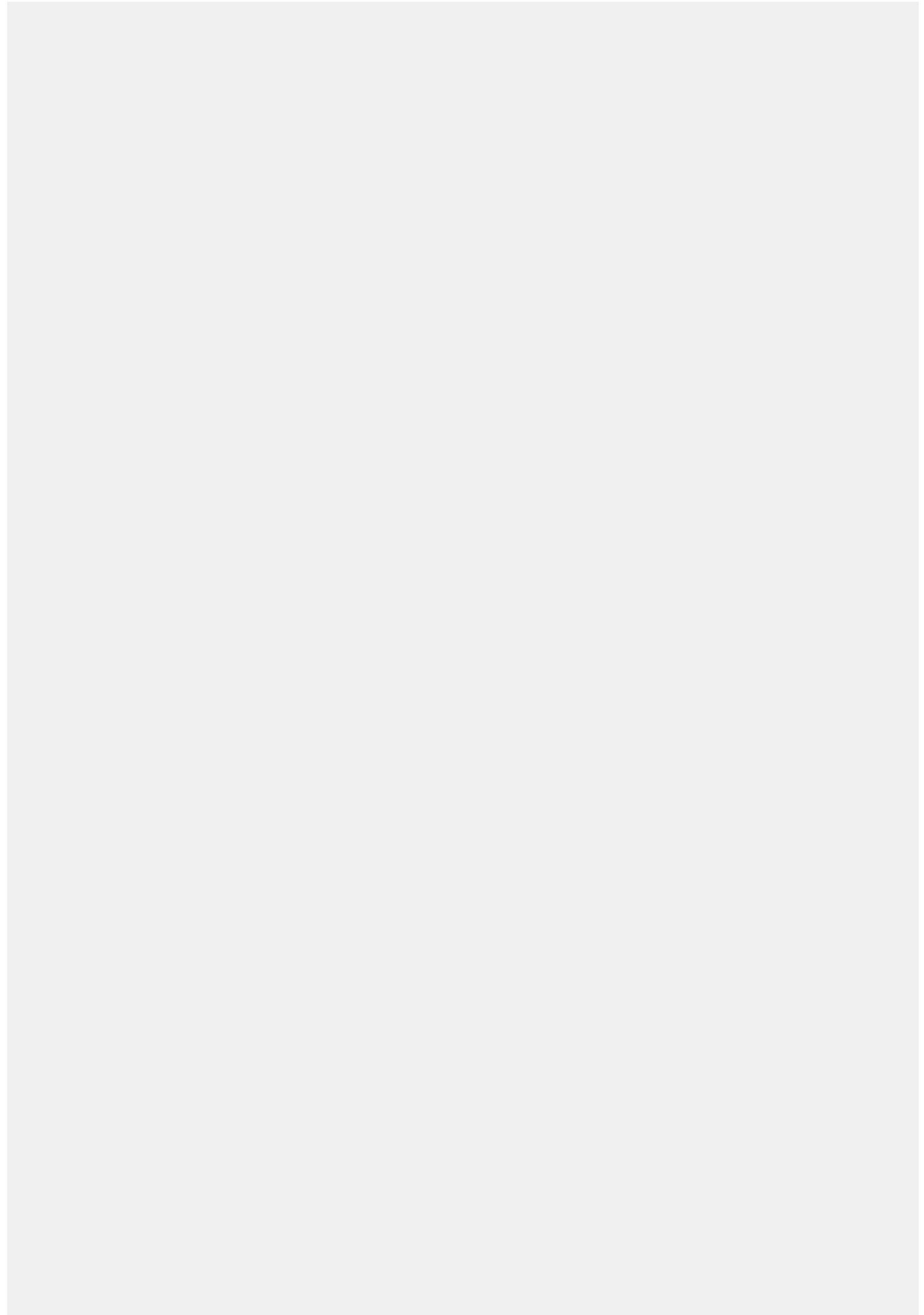


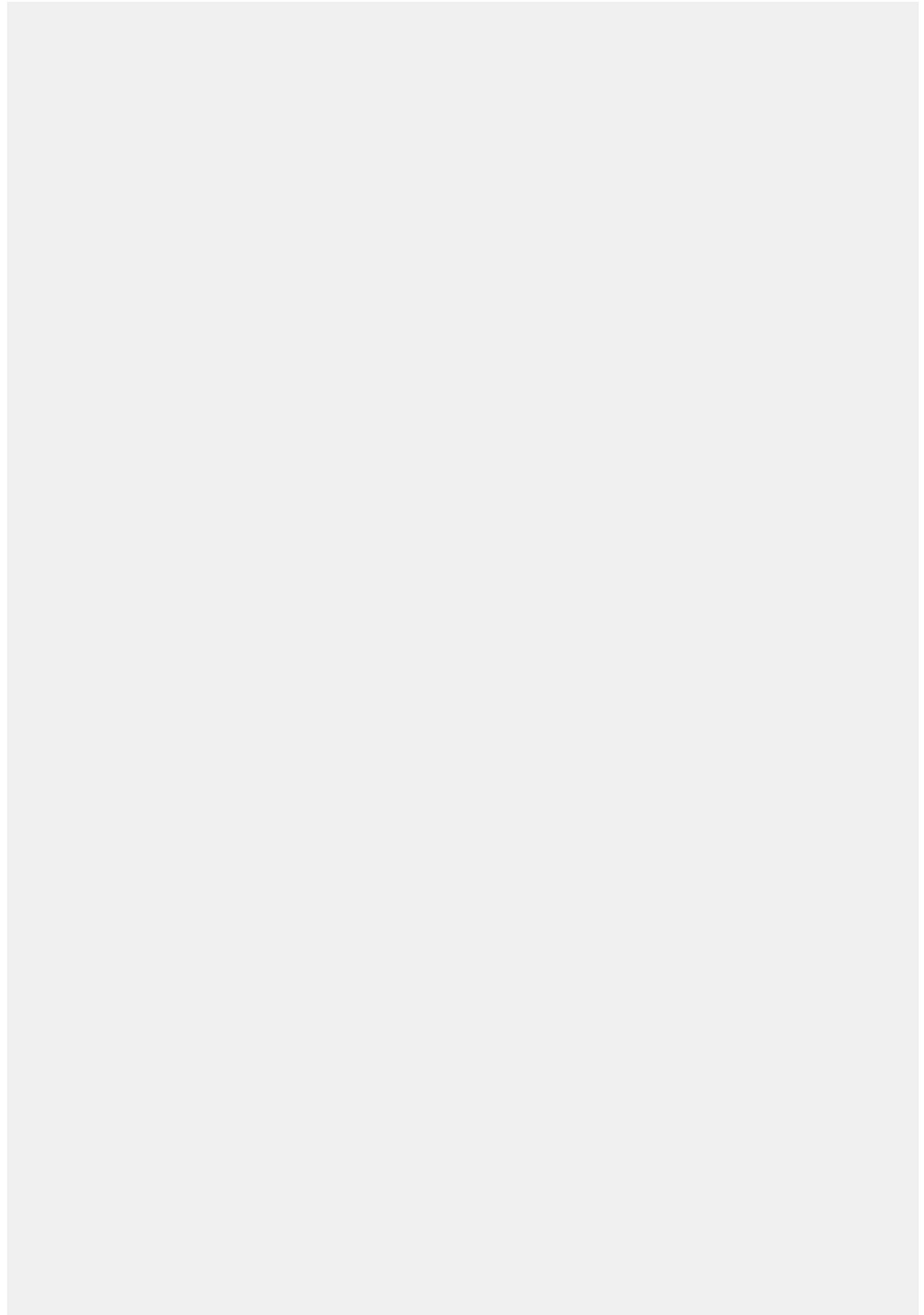












## ИНФОРМАЦИЯ

---

<b>Дата основания</b>	16 апреля 2003
<b>Местоположение</b>	Россия
<b>Сайт</b>	<a href="https://dsec.ru">dsec.ru</a>
<b>Численность</b>	51–100 человек
<b>Дата регистрации</b>	23 марта 2012

## ССЫЛКИ

---

[Сайт Digital Security](https://dsec.ru)  
dsec.ru

[Digital Security в Telegram](https://t.me/dsec)  
t.me

[Digital Security ВК](https://vk.com/dsec)  
vk.com

[Digital Security на YouTube](https://www.youtube.com/channel/UC...)  
www.youtube.com

[Блог Digital Security на VC](https://vc.ru/dsec)  
vc.ru

[Digital Security на Дзене](https://dsec.ru)

Digital Security на дзене

dzen.ru

Международная конференция ZeroNights

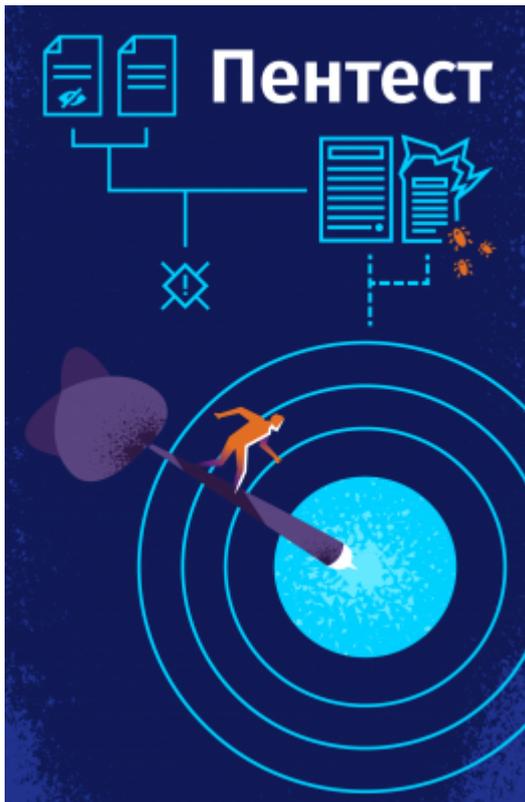
zeronights.ru

HeadHunter Digital Security

spb.hh.ru

---

## ВИДЖЕТ



---

## ВИДЖЕТ





---

## БЛОГ НА ХАБРЕ

---

вчера в 05:00

### Полевой набор пентестера

👁 8K    💬 15 +15

---

30 сентября в 09:55

### Фаззинг JS-движков с помощью Fuzzilli

👁 1.7K    💬 2 +2

---

13 сентября в 07:01

### Что мы используем для анализа Android-приложений

👁 3.8K    💬 3

---

7 сентября в 07:00

### Готовим Android к пентесту — WSA edition

👁 2.8K    💬 3 +3

---

31 августа в 04:00

### Атакуем синезубого короля

👁 7.2K    💬 22 +22

---