TECHDOM & CODE · CHAPTER 2

~$ open safe-design

Safe design

A tool that moves money inside a game is only safe if four things hold: a limit set in advance, a wait, a stop that works in every state, and a check from outside the page. This chapter explains each one and names the pressure designs to refuse.

ENTRY T2 CATEGORY TECHDOM & CODE CHECKED 3 NOVEMBER 2026
READING 9 MIN SECTIONS 6



ls cover/ T2.webp

EVIDENCE ◆ Verified ◆ Reported ◆ Vivian's field notes

This chapter is not advice for your situation.

IN 30 SECONDS

01 A safe tool has four parts: a limit, a wait, a stop and a trusted check.

02 A limit held only by the page is no limit. It has to live where the tool can't change it.

03 A stop must work in every state of the tool and must never cost anything.

04 Hidden totals, locks, paid exits, forced respins and sulking mascots are pressure designs to refuse.

05 Safe design lowers the risk. It doesn't remove it. Someone in a spiral needs the tool off and help.

YOU WILL LEARN

- ✓ The four-part rule and how to test each part.
- ✓ Why the limit and the running total must come from outside the tool.
- ✓ The pressure designs, how each one works on you, and what a safe tool does instead.
- ✓ What consent looks like inside a control program.
- ✓ How to run a small guard and read its real output.

> *I build tools too, so I know what these rules cost. Each one earns less money, which is why the scene rarely rewards them.* ◆ *I don't claim any tool of mine meets this list. Check mine against it like you'd check anyone's.*
>
> — VIVIAN

~/01-the-four-parts

The four parts

Every safe tool has the same four parts. The rest of this chapter is detail on them. ◆

`$ A limit set in advance`

The person who pays sets a ceiling while calm, before the session, and the tool can't raise it. ◆

`$ A wait`

A minimum time between two sends, and a longer wait before the limit itself can go up. ◆

`$ A stop that works in every state`

One visible control that works at once, asks nothing and costs nothing. ◆

`$ A trusted check outside the page`

The total comes from the payment provider or the bank, and the person accepts inside the program, never only on a web page. ◆

A visible total, a stop, a limit and a wait all cut the number of sends, and the most successful designs in the scene do the opposite. ◆

Safe design lowers risk. It doesn't remove it. A determined person can get past a blocking or draining extension, so a tool is never your only protection. ◆ If spending already feels hard to stop, the tool needs to be off and you need help: see The spiral and Where to get help.

These rules guard against the moment, whoever you are. One author compared some findom cases to problem gambling. ◆ A first-person account

raises a consent concern about a client taking on credit-card debt. ◆ Each rule below moves a decision out of the heated moment and into a calm one.

~/02-the-limit-and-the-wait

The limit and the wait

A limit that lives only in the page can be edited, reloaded or bypassed. The limit has to be read by the server, or better, from the payment provider itself. The daily and monthly limits the person chose are applied there. ◆

A limit typed in the middle of a session is weak. Set it while you're calm, before you start, and keep it somewhere you can't change it in the moment: a server that enforces it, your bank, or a person you trust. ◆

Count more than what's done. A limit that counts only finished sends can be overshot by a send already on its way, so a tool counts what's confirmed, what's owed and the next step. A limit is a ceiling, never a target: an item that would push the total over it never gets picked. ◆

Raise a limit only after a wait. Letting the person type any limit they like, with no maximum, is respectful but weak against an urge in the moment. A cooling-off delay before a limit goes up fixes that, and lowering a limit should always be instant. ◆

Your bank is a brake you can keep on. An unusual run of late-night card payments can get you a call from the bank, and a confirmation on your phone for each payment slows spending down. A version of a tool sold as getting round that confirmation removes a brake you chose. Refuse it. ◆

A tool that keeps your signed-in session and a saved card, and buys in the background, takes away the pause you'd normally have between decisions. The only brake left is the limit you set beforehand. ◆

A wait comes in three forms. One is a minimum time between two payments, enforced where the tool can't skip it. Another is an acknowledgement timer: a command not accepted within a short window never runs. The third is an unwatched session that ends by itself.

Small amounts at short intervals still add up. Do the maths: USD 1.00 every 6 seconds is USD 600.00 in one hour, which is why an overnight mode needs a limit.

~/03-a-stop-that-works-in-every-state

A stop that works in every state

The stop is visible, works at once, costs nothing and asks no question. Test it in every state: idle, waiting, in the middle of a payment, after a special mode, after an error. A build can lose its stop control after another control is pressed, so test the stop after every control. A drain you can't stop breaks the basic promise of techdom.

Closing the window is a last resort, and it doesn't count as a stop. Force-quitting always works, and you should know how to do it before you start, but a tool must not rely on it. On Windows you can remove an app from Settings, Apps, Installed apps. Some apps can't be removed from Settings, so find out how a tool is removed before you install it.

A good tool fails by stopping. When something's unclear, it never retries blindly.

- No saved card: the tool stops.
- A bank verification appears: the tool waits for you and doesn't click on.
- A payment isn't confirmed by the platform: it isn't counted.

- A payment went out without confirmation: the run stops and warns you.
- The page reloads: a stop stays a stop.

TRY IT

The stop does nothing

You pressed a special button in a tool, and now the stop button doesn't respond. The tool is still sending. What do you do?

A Press stop again and wait

Not this one. Waiting lets the sends go on. A stop that fails is a defect, and waiting won't fix it.

B Force the tool closed, remove it, check your order history, and tell the maker

The right answer. Right. Force-quit always works, and the maker needs to know the stop failed. See If you already sent.

C Raise your limit so the tool can finish

Not this one. A limit goes up only after a wait, and never to end a problem.

~/04-a-check-from-outside

A check from outside

The first meaning of a trusted check is the total. The amount counted comes from the payment provider's own record, which the tool can't write to. The

sub's browser never sets the price. A payment counts only when the processor confirms it. ◆

The second meaning is consent. The person accepts inside the program, and a controller can only ask. The web side can only make settings stricter. Everything starts switched off, with a required list of what's allowed. ◆

A quick exit is always there: a keyboard shortcut, a button and a tray menu. A safety word is agreed when consent is given, so a session can end in a second. ◆

Trust the design over the dom's word. Whoever runs a control tool can act on your accounts at any moment, and this wiki doesn't recommend control of accounts or machines in any design. Visible limits, an emergency stop and nothing that can act without you reduce that risk and never remove it. A dom who really uses consent tools shows them: a stop control, a record of what she can see, and a clear line between teasing and harm. ◆

The dom can't be your third party. She's paid by the sends, so she can stop, listen and point you elsewhere, but she can't manage your limits. Your bank, a person you trust, or a blocker you chose and can't easily switch off can fill that role. ◆

Keep secrets out of the tool. Keys and tokens go into a masked prompt in the owner's own terminal, never into a chat, a commit or a screenshot. A safe tool holds no names, email addresses or card numbers, and any identifier it creates is random, never built from the computer name or the Windows account name. ◆

The example below is a small guard written in plain JavaScript. It sends nothing. Before each send it answers one question: may this go ahead?

```
# A guard with all four parts
```

config.json

```
{
  "limitCents": 10000,
  "waitSeconds": 900,
  "raiseWaitSeconds": 86400
}
```

example.js

```
"use strict";

const { createGuard } = require("./guard");
const config = require("./config.json");

const MINUTE = 60;
const HOUR = 3600;

function usd(cents) {
  return "USD " + (cents / 100).toFixed(2);
}

// A stand-in for the outside source, such as the payment
provider's own record.
// The guard can read it. Nothing in guard.js can write to it.
function makeProvider(startCents = 0) {
  let confirmed = startCents;
  return {
    read: () => confirmed,
    confirm: (cents) => {
      confirmed += cents;
    },
  };
}

function makeGuard(provider, overrides = {}) {
  return createGuard({ ...config, readConfirmedCents:
provider.read, ...overrides });
```

```
}

function describe(result) {
  if (result.ok) return "allowed";
  return "refused (" + result.reason + (result.secondsLeft ?
", " + result.secondsLeft + " s left" : "") + ")";
}

console.log("1. The limit and the wait. Limit " +
usd(config.limitCents) + ", wait " + config.waitSeconds /
MINUTE + " minutes.");
{
  const provider = makeProvider();
  const guard = makeGuard(provider);
  const attempts = [
    [0, 2500],
    [1 * MINUTE, 2500],
    [16 * MINUTE, 2500],
    [32 * MINUTE, 2500],
    [48 * MINUTE, 2500],
    [64 * MINUTE, 2500],
  ];
  for (const [at, amount] of attempts) {
    const result = guard.ask(amount, at);
    console.log("  minute " + String(at / MINUTE).padStart(2)
+ ": ask " + usd(amount) + " -> " + describe(result));
    if (result.ok) {
      provider.confirm(amount);
      guard.release(amount);
    }
  }
  const s = guard.status(64 * MINUTE);
  console.log("  running total " + usd(s.confirmedCents) + "
of " + usd(s.limitCents));
}

console.log("2. The tool counts wrong, the outside record does
not.");
{
  const provider = makeProvider(9000);
```

```
  const guard = makeGuard(provider);
  const toolCounter = 0;
  console.log("  the tool says it has sent " +
usd(toolCounter));
  console.log("  the outside record says " +
usd(provider.read()));
  console.log("  ask " + usd(2500) + " -> " +
describe(guard.ask(2500, 0)));
}

console.log("3. A raise waits, a lowering does not.");
{
  const provider = makeProvider();
  const guard = makeGuard(provider);
  console.log("  limit now " +
usd(guard.status(0).limitCents));
  console.log("  ask to raise to " + usd(30000) + " -> " +
guard.setLimit(30000, 0).effective);
  console.log("  after 1 hour the limit is " +
usd(guard.status(HOUR).limitCents));
  console.log("  after 23 hours the limit is " +
usd(guard.status(23 * HOUR).limitCents));
  console.log("  after 24 hours the limit is " +
usd(guard.status(24 * HOUR).limitCents));
  console.log("  ask to lower to " + usd(5000) + " -> " +
guard.setLimit(5000, 25 * HOUR).effective);
  console.log("  the limit is " + usd(guard.status(25 *
HOUR).limitCents));
}

console.log("4. The stop, tried in every state.");
{
  const states = [
    ["idle", () => ({ guard: makeGuard(makeProvider()), now: 0
})],
    [
      "cooling down",
      () => {
        const guard = makeGuard(makeProvider());
        guard.ask(2500, 0);
```

```
        return { guard, now: 1 * MINUTE };
      },
    ],
    [
      "raise waiting",
      () => {
        const guard = makeGuard(makeProvider());
        guard.setLimit(30000, 0);
        return { guard, now: 1 * HOUR };
      },
    ],
    [
      "send reserved",
      () => {
        const guard = makeGuard(makeProvider());
        guard.ask(2500, 0);
        return { guard, now: 2 * HOUR };
      },
    ],
    ["limit reached", () => ({ guard:
makeGuard(makeProvider(10000)), now: 0 })],
    [
      "outside record broken",
      () => ({
        guard: makeGuard(makeProvider(), {
          readConfirmedCents: () => {
            throw new Error("offline");
          },
        }),
        now: 0,
      }),
    ],
  ];
  for (const [name, build] of states) {
    const { guard, now } = build();
    const before = describe(guard.ask(100, now));
    guard.stop();
    const after = describe(guard.ask(100, now + 48 * HOUR));
    console.log("  " + name.padEnd(22) + "before: " +
before.padEnd(38) + "after stop: " + after);
```

```
    }
  }
```

guard.js

```
"use strict";

// guard.js: the four-part rule in one small file.
//   1. A limit set in advance.
//   2. A wait between two sends, and a wait before the limit
can go up.
//   3. A stop that works in every state.
//   4. A check outside the tool: the total comes from a source
the tool cannot write.
// Money is whole cents and time is whole seconds. Nothing here
touches a network or a payment.

function isCents(n) {
  return Number.isInteger(n) && n >= 0;
}

function createGuard({ limitCents, waitSeconds,
raiseWaitSeconds, readConfirmedCents }) {
  if (!isCents(limitCents) || limitCents === 0) throw new
RangeError("limitCents must be a whole number above zero");
  if (!isCents(waitSeconds) || !isCents(raiseWaitSeconds))
throw new RangeError("waits must be whole seconds");
  if (typeof readConfirmedCents !== "function") throw new
TypeError("an outside total is required");

  const state = { limit: limitCents, raise: null, lastSendAt:
null, reserved: 0, stopped: false };

  function applyRaise(now) {
    if (state.raise && now >= state.raise.at) {
      state.limit = state.raise.limit;
      state.raise = null;
    }
  }

```

```
  function outsideTotal() {
    try {
      const value = readConfirmedCents();
      return isCents(value) ? value : null;
    } catch (error) {
      return null;
    }
  }

  function refuse(reason, extra) {
    return Object.assign({ ok: false, reason }, extra);
  }

  return {
    // Ask before every send. A yes reserves the amount;
nothing is sent by this file.
    ask(amountCents, now) {
      if (state.stopped) return refuse("stopped");
      if (!Number.isInteger(amountCents) || amountCents <= 0)
return refuse("bad-amount");
      applyRaise(now);
      const confirmed = outsideTotal();
      if (confirmed === null) return refuse("outside-total-
unreadable");
      if (confirmed + state.reserved + amountCents >
state.limit) return refuse("limit");
      if (state.lastSendAt !== null && now - state.lastSendAt <
waitSeconds) {
        return refuse("wait", { secondsLeft: waitSeconds - (now
- state.lastSendAt) });
      }
      state.reserved += amountCents;
      state.lastSendAt = now;
      return { ok: true };
    },

    // Call once the outside source shows the payment. Until
then the amount counts twice, which errs on the safe side.
    release(amountCents) {
      state.reserved = Math.max(0, state.reserved -
```

```
amountCents);
    },

    // Lowering the limit is immediate. Raising it only starts
a clock.
    setLimit(newLimitCents, now) {
      applyRaise(now);
      if (!isCents(newLimitCents) || newLimitCents === 0)
return { ok: false, reason: "bad-limit" };
      if (newLimitCents <= state.limit) {
        state.limit = newLimitCents;
        state.raise = null;
        return { ok: true, effective: "now" };
      }
      state.raise = { limit: newLimitCents, at: now +
raiseWaitSeconds };
      return { ok: true, effective: "after-wait", at:
state.raise.at };
    },

    // No argument, no question, no way to fail. It also
cancels any raise that is waiting.
    stop() {
      state.stopped = true;
      state.raise = null;
    },

    // The running total, always readable.
    status(now) {
      applyRaise(now);
      const confirmed = outsideTotal();
      return {
        limitCents: state.limit,
        confirmedCents: confirmed,
        reservedCents: state.reserved,
        stopped: state.stopped,
        raiseWaiting: state.raise !== null,
      };
    },
  };
```

```
  }

  module.exports = { createGuard };
```

```
output  $ node example.js
1. The limit and the wait. Limit USD 100.00, wait 15 minutes.
  minute  0: ask USD 25.00 -> allowed
  minute  1: ask USD 25.00 -> refused (wait, 840 s left)
  minute 16: ask USD 25.00 -> allowed
  minute 32: ask USD 25.00 -> allowed
  minute 48: ask USD 25.00 -> allowed
  minute 64: ask USD 25.00 -> refused (limit)
  running total USD 100.00 of USD 100.00
2. The tool counts wrong, the outside record does not.
  the tool says it has sent USD 0.00
  the outside record says USD 90.00
  ask USD 25.00 -> refused (limit)
3. A raise waits, a lowering does not.
  limit now USD 100.00
  ask to raise to USD 300.00 -> after-wait
  after 1 hour the limit is USD 100.00
  after 23 hours the limit is USD 100.00
  after 24 hours the limit is USD 300.00
  ask to lower to USD 50.00 -> now
  the limit is USD 50.00
4. The stop, tried in every state.
  idle                  before: allowed
after stop: refused (stopped)
  cooling down          before: refused (wait, 840 s left)
after stop: refused (stopped)
  raise waiting         before: allowed
after stop: refused (stopped)
  send reserved         before: allowed
after stop: refused (stopped)
  limit reached         before: refused (limit)
after stop: refused (stopped)
  outside record broken before: refused (outside-total-unreadable)
after stop: refused (stopped)
```

Real output, kept next to the code. First run: four sends of USD 25.00 are allowed at minutes 0, 16, 32 and 48. The ask at minute 1 is refused for the wait, with 840 seconds left, and the ask at minute 64 is refused for the limit. Second run: the tool counts USD 0.00 while the outside record says USD 90.00, and a USD 25.00 ask is refused. Third run: a raise to USD 300.00 waits, with the limit still at USD 100.00 after 23 hours and at USD 300.00 after 24 hours, while a lowering to USD 50.00 applies at once. Fourth run: in six states the stop turns every later ask into a refusal.

Read the output with the four parts in mind. The first run shows the limit and the wait holding. The second shows the outside record winning over the tool's own count. The third shows a raise waiting and a lowering going through at once. The fourth shows the stop refusing everything, even when the outside record is broken, because an unreadable total makes the guard stop instead of guess.

This guard has its weak spots. The provider is a stand-in inside the file. A guard only counts as an outside check when it runs somewhere the tool can't edit, and a guard in the same page as the tool can be changed by that page. ◆ Guards get tested by breaking them on purpose, as Code tutorials shows.

`~/05-pressure-designs-to-refuse`

Pressure designs to refuse

A pressure design makes the next payment harder to refuse. A safe tool does the opposite, and a maker who copies these is selling a trap with a theme on it. ◆

Pressure designs, and the answer

WHAT THEY SAY OR DO

01 A total you can't see: only a multiplier such as 10x, or random prices

THE COUNTER

A vague chance of a cheap pull feels better than a clear price, even when the clear price is lower. Treat any tool that never shows the total before the charge as a warning, and set your limit first. A safe tool shows a fixed price. ◆

WHAT THEY SAY OR DO

02 A lock you can't undo, or an exit that costs money, such as a code sold to reopen a locked phone

THE COUNTER

With no proper stop, force-quit is your only way out, and a paid exit makes money from the lock. Refuse both. A safe tool has one visible, free way out. ◆

WHAT THEY SAY OR DO

03 A forced respin when a result is low, with a voice line saying it's not enough

THE COUNTER

A random result becomes a guilt trigger. Treat it as pressure, whatever the game calls it. ◆

WHAT THEY SAY OR DO

04 A sentence you must type before the next spin

THE COUNTER

Friction that deepens the scene also makes quitting feel like failing. A safe tool never makes you do a ritual to continue. ◆

WHAT THEY SAY OR DO

05 A mascot that sulks, nags, leaves when you go idle or moves the stop button

THE COUNTER

A character you feel for can steer your spending, and that's what to watch for. Effects must never cover the stop or the total. ◆

WHAT THEY SAY OR DO

06 A special mode with higher minimums, or a surprise mode switched on only after a few sends

THE COUNTER

The biggest cost arrives once you're already into it. Read every mode before you start and set your limit first. ◆

WHAT THEY SAY OR DO

07 Cards or rewards that fade unless you keep paying

THE COUNTER

That's a hidden subscription. The price you see is low and the real cost is higher. A safe tool states the full cost. ◆

WHAT THEY SAY OR DO

08 Time-limited banners, or progress stored only inside the tool

THE COUNTER

A deadline on a decision, and a collection that vanishes when you uninstall, push you to keep the tool installed. A safe tool keeps progress somewhere it lasts and lets you leave. ◆

WHAT THEY SAY OR DO

09 A public tally of sends, or a message in a server each time you run the tool

THE COUNTER

Nobody wants to be seen quitting, so being watched makes stopping harder, and it's also a record of your habit. A safe tool keeps your total private. ◆

WHAT THEY SAY OR DO

10 An overnight mode or a long unattended session

THE COUNTER

Small amounts at short intervals add up while you sleep. A safe tool ends an unwatched session. ◆

WHAT THEY SAY OR DO

11 A prompt that targets your payday, or shows up when your balance is low

THE COUNTER

A tool aimed at your cash flow is aimed at your limits. When a sub says money is short, the money part stops for that evening. ◆

WHAT THEY SAY OR DO

12 A chain of click-to-post links, each drafting a post for you with a harsher label

THE COUNTER

Each step raises the commitment. You publish the text on your own account, so read every draft first and don't use an account tied to your real identity. ◆

Each line is written for the person who has to recognise and refuse the design.

> // Vivian's field notes
>
> *In my experience, the tools that spread fastest are the ones with no visible total and no limit. If a maker ships a stop, a limit and a running total without being asked, that tells you more about respect than any label on the page.*
>
> ◆

- `A limit on the page is a limit.`
+ The page can be edited or reloaded. A limit only counts where the tool can't change it. ◆

- `A fair game is a safe game.`
+ A fair game has code that matches its stated odds. A game can match its odds exactly and still be built to take money, so safe means you can stop and you can see the total. A game with fictional chips has no money to manage while the chips stay fictional. ◆

- `A friendly label proves a tool is safe.`
+ Hidden or scrambled code and a reassuring name prove nothing. Look for a stop, a total on screen and a record of what the tool can see. ◆

`~/06-before-a-tool-reaches-a-person`

Before a tool reaches a person

A wrong build can charge a real person a different amount than the wheel shows, and a test switch that skips the payment page, left in a build, is a risk.

◆ These steps are for makers, and they're also a list of questions for anyone about to install.

```
# A release routine
```

a-release-routine.md · 6 lines copy

```
1 Label every build test or real. A test build that really takes
  money must say so.
2 Remove every switch that skips the payment page.
3 Check the file you actually ship by comparing its size or hash
  with the one you tested.
4 Show the amount displayed next to the amount charged.
5 Rehearse end to end: a fake provider first, the provider's test
  mode next, and one tiny agreed real payment last.
6 If the tool misbehaves, pull it from live sessions until it's
  fixed and retested.
```

Effects can hurt the body too. The W3C success criterion 2.3.1 says pages must not flash more than three times in any one-second period, unless the flash stays below set thresholds. ◆ Flashing content can cause migraines, dizziness, nausea and seizures. ◆ The seam where a loop restarts counts toward the limit. ◆

A tool should honour the reduced-motion setting, which tells it whether the user asked the device to cut non-essential motion. ◆ The stop cuts every effect within one frame, and the controls stay visible and clickable under every effect. Measure the flash rate. Don't judge it by eye. ◆

TRY IT

Check yourself

QUESTION 1 OF 3

Where should the running total come from?

- ◯ The tool's own counter
- ◯ The payment provider's record, which the tool can't write to
- ◯ Your memory of the evening

Answer. The payment provider's record, which the tool can't write to. A total held by the tool can be wrong, edited or reset. The tool can't change an outside record.

QUESTION 2 OF 3

When can a limit go up?

- ◯ Right away, when you ask
- ◯ When the tool suggests it
- ◯ Only after a wait, while lowering it stays instant

Answer. Only after a wait, while lowering it stays instant. A raise made in the heat of the moment is the weak point. A wait moves that decision to a calm time.

QUESTION 3 OF 3

A tool offers a top tier that locks its master switch. What does safe design say?

- ◯ It’s fine, because uninstalling always works
- ◯ Refuse it, because a safe tool keeps one visible, free way out
- ◯ It’s fine if the price is shown

Answer. Refuse it, because a safe tool keeps one visible, free way out. A lock leaves force-quit as the only way out. A tool must not depend on that.

REMEMBER IT

- ♥ Four parts: a limit set in advance, a wait, a stop that works in every state, and a check outside the page.
- ♥ A limit held by the page is no limit, and a total counted by the tool isn’t a total.
- ♥ Raise a limit only after a wait. Lower it at once.
- ♥ Refuse locks, paid exits, hidden totals, forced respins and sulking mascots.
- ♥ A safe tool lowers the risk. If spending is hard to stop, switch it off and get help.

SOURCES AND EVIDENCE

Check it

Every claim with its evidence level and its sources.

EVIDENCE LEVELS

◆ Verified	Checked against the original source.
◆ Reported	Stated by a serious secondary source read in full.
◆ Community	Widely said in the scene; no document proves it.
◆ Vivian's field notes	First-hand experience, not a study.
◆ Unverified	Plausible, not proven yet.

01 Each one earns less money, which is why the scene rarely rewards them. ◆

02 The rest of this chapter is detail on them. ◆

03 The person who pays sets a ceiling while calm, before the session, and the tool can't raise it. ◆

04 A minimum time between two sends, and a longer wait before the limit itself can go up. ◆

05 One visible control that works at once, asks nothing and costs nothing. ◆

06 The total comes from the payment provider or the bank, and the person accepts inside the program, never only on a web page.

◆

07 A visible total, a stop, a limit and a wait all cut the number of sends, and the most successful designs in the scene do the opposite.

◆

08 A determined person can get past a blocking or draining extension, so a tool is never your only protection.

◆

09 One author compared some findom cases to problem gambling.

◆

The Psychology of Financial Dominatrixes (blog post) (https://www.psychologytoday.com/us/blog/in-excess/201612/the-psychology-financial-dominatrixes)

Accessed 3 November 2026

10 A first-person account raises a consent concern about a client taking on credit-card debt.

◆

Diary of a Financial Dominatrix, part one (https://www.theskinny.co.uk/sexuality/deviance/diary-of-a-financial-dominatrix-part-one)

Archive (https://web.archive.org/web/20200805021033/https://www.theskinny.co.uk/sexuality/deviance/diary-of-a-financial-dominatrix-part-one)

Accessed 3 November 2026

11 The limit has to be read by the server, or better, from the payment provider itself.

◆

12 The daily and monthly limits the person chose are applied there.

◆

13 Set it while you're calm, before you start, and keep it somewhere you can't change it in the moment: a server that enforces it, your bank, or a person you trust.

◆

14 A limit that counts only finished sends can be overshot by a send already on its way, so a tool counts what's confirmed, what's owed and the next step.

◆

15 A limit is a ceiling, never a target: an item that would push the total over it never gets picked.

◆

16 A cooling-off delay before a limit goes up fixes that, and lowering a limit should always be instant.

◆

17 An unusual run of late-night card payments can get you a call from the bank, and a confirmation on your phone for each payment slows spending down.

◆

18 Refuse it.

◆

19 A tool that keeps your signed-in session and a saved card, and buys in the background, takes away the pause you'd normally have between decisions.

◆

20 The only brake left is the limit you set beforehand.

◆

21 The third is an unwatched session that ends by itself.

◆

22 Do the maths: USD 1.00 every 6 seconds is USD 600.00 in one hour, which is why an overnight mode needs a limit.

◆

23 The stop is visible, works at once, costs nothing and asks no question.

◆

24 A build can lose its stop control after another control is pressed, so test the stop after every control.

◆

25 A drain you can't stop breaks the basic promise of techdom.

◆

26 Force-quitting always works, and you should know how to do it before you start, but a tool must not rely on it.

◆

27 On Windows you can remove an app from Settings, Apps, Installed apps.

◆

Uninstall or remove apps and programs in Windows | Microsoft Support (https://support.microsoft.com/en-us/windows/uninstall-or-remove-apps-and-programs-in-windows-4b55f974-2cc6-2d2b-d092-5905080eaf98)

Archive (https://web.archive.org/web/20261024175047/https://support.microsoft.com/en-us/windows/uninstall-or-remove-apps-and-programs-in-windows-4b55f974-2cc6-2d2b-d092-5905080eaf98)

Accessed 3 November 2026

28 Some apps can't be removed from Settings, so find out how a tool is removed before you install it.

◆

Uninstall or remove apps and programs in Windows | Microsoft Support (https://support.microsoft.com/en-us/windows/uninstall-or-remove-apps-and-programs-in-windows-4b55f974-2cc6-2d2b-d092-5905080eaf98)

Archive (https://web.archive.org/web/20261024175047/https://support.microsoft.com/en-us/windows/uninstall-or-remove-apps-and-programs-in-windows-4b55f974-2cc6-2d2b-d092-5905080eaf98)

Accessed 3 November 2026

29 When something's unclear, it never retries blindly.

◆

30 No saved card: the tool stops.

◆

31 A bank verification appears: the tool waits for you and doesn't click on.

◆

32 A payment isn't confirmed by the platform: it isn't counted.

◆

33 A payment went out without confirmation: the run stops and warns you.

◆

34 The page reloads: a stop stays a stop.

◆

35 A payment counts only when the processor confirms it.

◆

36 The web side can only make settings stricter.

◆

37 Everything starts switched off, with a required list of what's allowed.

◆

38 A quick exit is always there: a keyboard shortcut, a button and a tray menu.

◆

39 A safety word is agreed when consent is given, so a session can end in a second.

◆

40 Visible limits, an emergency stop and nothing that can act without you reduce that risk and never remove it.

◆

41 A dom who really uses consent tools shows them: a stop control, a record of what she can see, and a clear line between teasing and harm.

◆

42 She’s paid by the sends, so she can stop, listen and point you elsewhere, but she can’t manage your limits.

◆

43 Your bank, a person you trust, or a blocker you chose and can’t easily switch off can fill that role.

◆

44 Keys and tokens go into a masked prompt in the owner’s own terminal, never into a chat, a commit or a screenshot.

◆

45 A safe tool holds no names, email addresses or card numbers, and any identifier it creates is random, never built from the computer name or the Windows account name.

◆

46 A guard only counts as an outside check when it runs somewhere the tool can’t edit, and a guard in the same page as the tool can be changed by that page.

◆

47 A safe tool does the opposite, and a maker who copies these is selling a trap with a theme on it.

◆

48 A safe tool shows a fixed price.

◆

49 A safe tool has one visible, free way out.

◆

50 Treat it as pressure, whatever the game calls it.

◆

51 A safe tool never makes you do a ritual to continue.

◆

52 Effects must never cover the stop or the total.

◆

53 Read every mode before you start and set your limit first.

◆

54 A safe tool states the full cost.

◆

55 A safe tool keeps progress somewhere it lasts and lets you leave.

◆

56 A safe tool keeps your total private.

◆

57 A safe tool ends an unwatched session.

◆

58 When a sub says money is short, the money part stops for that evening.

◆

59 You publish the text on your own account, so read every draft first and don’t use an account tied to your real identity.

◆

60 In my experience, the tools that spread fastest are the ones with no visible total and no limit. If a maker ships a stop, a limit and a running total without being asked, that tells you more about respect than any label on the page.

◆

61 A limit only counts where the tool can't change it.

◆

62 A game with fictional chips has no money to manage while the chips stay fictional.

◆

63 Look for a stop, a total on screen and a record of what the tool can see.

◆

64 A wrong build can charge a real person a different amount than the wheel shows, and a test switch that skips the payment page, left in a build, is a risk.

◆

65 The W3C success criterion 2.3.1 says pages must not flash more than three times in any one-second period, unless the flash stays below set thresholds.

◆

Web Content Accessibility Guidelines (WCAG) 2.2 (https://www.w3.org/TR/WCAG22/)

Archive (https://web.archive.org/web/20261029121709/https://www.w3.org/TR/WCAG22/)

Accessed 3 November 2026

66 Flashing content can cause migraines, dizziness, nausea and seizures.

◆

Understanding Success Criterion 2.3.1: Three Flashes or Below Threshold | WAI | W3C (https://www.w3.org/WAI/WCAG22/Understanding/three-flashes-or-below-threshold.html)

Archive (https://web.archive.org/web/20261001165853/https://www.w3.org/WAI/WCAG22/Understanding/three-flashes-or-below-threshold.html)

Accessed 3 November 2026

67 The seam where a loop restarts counts toward the limit.

◆

68 A tool should honour the reduced-motion setting, which tells it whether the user asked the device to cut non-essential motion.

◆

prefers-reduced-motion CSS media feature - CSS | MDN (https://developer.mozilla.org/en-US/docs/Web/CSS/Reference/At-rules/@media/prefers-reduced-motion)

Archive (https://web.archive.org/web/20261031201800/https://developer.mozilla.org/en-US/docs/Web/CSS/Reference/At-rules/%40media/prefers-reduced-motion)

Accessed 3 November 2026

69 The stop cuts every effect within one frame, and the controls stay visible and clickable under every effect.

◆

70 Don’t judge it by eye.

◆