```
/**
 * ============================================================
 * [BRAND NAME] — Security ROI Calculator
 * Formula Logic Module  |  Version 1.0  |  July 2026
 * ============================================================
 *
 * HOW TO USE THIS FILE
 * --------------------
 * 1. Include this file before your main calculator script, OR
 *    copy the contents into your existing rosi-calculator.html
 *    inside the <script> tag, replacing the matching sections.
 *
 * 2. Call  recalculate()  any time a user changes an input.
 *    It reads all inputs, runs every formula, and returns a
 *    results object you can use to update the UI.
 *
 * 3. A worked example with expected outputs is at the bottom
 *    of this file. Use it to verify your implementation.
 *
 * 4. Every formula has a plain-English comment above it.
 *    If a number looks arbitrary, the comment explains where
 *    it came from.
 * ============================================================
 */


// ============================================================
// SECTION 1 — LOOKUP TABLES
// All the reference data the formulas pull from.
// ============================================================

/**
 * SECTOR BASE COSTS
 * Source: IBM X-Force Threat Intelligence Index 2026.
 * baseCost  = full average breach cost for a large organisation in that sector.
 * breachProb = annual probability of a breach occurring (0–1 scale).
 */
var SECTOR_DATA = {
  healthcare:    { baseCost: 11200000, breachProb: 0.32, label: 'Healthcare' },
  finance:       { baseCost:  6080000, breachProb: 0.28, label: 'Financial Servic
  insurance:     { baseCost:  5440000, breachProb: 0.26, label: 'Insurance' },
  manufacturing: { baseCost:  5560000, breachProb: 0.30, label: 'Manufacturing' }
  retail:        { baseCost:  3480000, breachProb: 0.24, label: 'Retail' },
  government:    { baseCost:  4120000, breachProb: 0.22, label: 'Government' },
```

```
  education:     { baseCost:  3400000, breachProb: 0.20, label: 'Education' },
  technology:    { baseCost:  4960000, breachProb: 0.26, label: 'Technology' }
};

/**
 * ORGANISATION SIZE MULTIPLIERS
 * Scales the sector base cost to the user's organisation size.
 * Source: IBM breach cost scaling analysis.
 */
var SIZE_MULT = {
  small:      0.35,   // < 250 employees
  mid:        0.70,   // 250–2,500  (default)
  large:      1.00,   // 2,500–10,000  (reference scale)
  enterprise: 1.65    // 10,000+
};

/**
 * REGULATORY FINE DATA
 * maxFine    = the hard ceiling on fines under that framework (USD).
 * pctRevenue = the % of annual revenue that can be fined.
 * The calculator uses whichever is LOWER — this prevents
 * over-stating fines for very large enterprises.
 * Sources: GDPR Article 83, HIPAA enforcement, DORA Article 35,
 *          NY DFS Oct 2025 enforcement, NAIC MDL-668, CMMC 2.0.
 */
var REG_DATA = {
  gdpr:  { maxFine: 20000000, pctRevenue: 0.04, label: 'GDPR / UK GDPR' },
  hipaa: { maxFine:  1900000, pctRevenue: 0.01, label: 'HIPAA / HITECH' },
  dora:  { maxFine: 10000000, pctRevenue: 0.02, label: 'DORA' },
  nyfs:  { maxFine:  5000000, pctRevenue: 0.015,label: 'NY DFS Part 500' },
  naic:  { maxFine:  2000000, pctRevenue: 0.01, label: 'NAIC MDL-668' },
  cmmc:  { maxFine:   500000, pctRevenue: 0.005,label: 'CMMC 2.0' },
  none:  { maxFine:        0, pctRevenue: 0,    label: 'None' }
};

/**
 * SECURITY CONTROLS
 * riskReduction = the fraction by which this control reduces annual loss
 *                 when active. Values are additive, capped at 0.88 total.
 * Sources: IBM, Microsoft, Veeam, Verizon DBIR 2026, NIST.
 * See formula reference PDF for full justification of each coefficient.
 */
var CONTROLS = [
  { id: 'mfa',      label: 'Multi-Factor Authentication',      riskReduction: 0.2
  { id: 'edr',      label: 'Endpoint Detection & Response',    riskReduction: 0.2
  { id: 'ztna',     label: 'Zero Trust / Network Segmentation',riskReduction: 0.1
  { id: 'backup',   label: 'Tested Air-Gapped Backups',        riskReduction: 0.2
```

```
  { id: 'siem',     label: 'SIEM / Security Monitoring',       riskReduction: 0.1
  { id: 'pam',      label: 'Privileged Access Management',     riskReduction: 0.1
  { id: 'ir',       label: 'Incident Response Plan (Tested)',  riskReduction: 0.1
  { id: 'training', label: 'Security Awareness Training',      riskReduction: 0.1
  { id: 'vuln',     label: 'Vulnerability Management',         riskReduction: 0.1
  { id: 'dlp',      label: 'Data Loss Prevention',             riskReduction: 0.1
];

/**
 * THREAT VECTORS
 * aleFactor = how much this threat type contributes to the overall
 *             threat multiplier when enabled. Values are additive.
 * Sources: IBM X-Force 2026, Verizon DBIR 2026, WEF 2026.
 */
var THREATS = [
  { id: 'ransomware', label: 'Ransomware',         aleFactor: 0.65, defaultOn: tr
  { id: 'phishing',   label: 'Phishing / BEC',     aleFactor: 0.55, defaultOn: tr
  { id: 'supply',     label: 'Supply Chain',        aleFactor: 0.45, defaultOn: t
  { id: 'insider',    label: 'Insider Threat',      aleFactor: 0.35, defaultOn: f
  { id: 'cloud',      label: 'Cloud Misconfiguration',aleFactor: 0.30, defaultOn:
  { id: 'ddos',       label: 'DDoS Attack',         aleFactor: 0.20, defaultOn: f
];

/**
 * CURRENCY CONVERSION RATES (relative to USD)
 * Update these periodically. Rates as of July 2026.
 */
var CURRENCY = {
  USD: { symbol: '$',  rate: 1.00 },
  GBP: { symbol: '£',  rate: 0.79 },
  EUR: { symbol: '€',  rate: 0.92 },
  AUD: { symbol: 'A$', rate: 1.53 }
};


// ============================================================
// SECTION 2 — THE MAIN CALCULATION FUNCTION
// Call this whenever any input changes.
// Pass in all the user's inputs as a plain object.
// Get back all the results as a plain object.
// ============================================================

/**
 * calculate(inputs)
 *
 * @param {Object} inputs
 *   sector        {string}  — key from SECTOR_DATA
```

```
 *   orgSize       {string}  — key from SIZE_MULT
 *   revenue       {number}  — annual revenue in USD
 *   secBudget     {number}  — annual security budget in USD
 *   records       {number}  — number of records at risk
 *   priorIncidents{number}  — number of prior breaches (last 24 months)
 *   regulation    {string}  — key from REG_DATA
 *   vendors       {number}  — number of high-access suppliers
 *   activeControls{string[]}— array of control IDs that are ON
 *   activeThreats {string[]}— array of threat IDs that are ON
 *   currency      {string}  — key from CURRENCY (default 'USD')
 *   // Expert mode inputs (all have defaults if omitted):
 *   downtimeCostPerHour {number} — default 25000
 *   downtimeHours       {number} — default 72
 *   churnRatePct        {number} — default 15 (percent, not decimal)
 *   customerLTV         {number} — default 5000
 *   insurancePremium    {number} — default 75000
 *   ipValue             {number} — default 2000000
 *
 * @returns {Object} results — see end of function for full shape
 */
function calculate(inputs) {

  // --- Read inputs, apply defaults ----------------------------------

  var sector     = inputs.sector     || 'technology';
  var orgSize    = inputs.orgSize    || 'mid';
  var revenue    = inputs.revenue    || 0;
  var secBudget  = inputs.secBudget  || 0;
  var records    = inputs.records    || 0;
  var incidents  = inputs.priorIncidents || 0;
  var regulation = inputs.regulation || 'none';
  var vendors    = inputs.vendors    || 0;
  var currency   = inputs.currency   || 'USD';

  var activeControls = inputs.activeControls || [];
  var activeThreats  = inputs.activeThreats  || [];

  // Expert mode — use provided value or fall back to sensible default
  var dtCostPerHr = inputs.downtimeCostPerHour || 25000;
  var dtHours     = inputs.downtimeHours       || 72;
  var churnRate   = (inputs.churnRatePct || 15) / 100;  // convert % to decimal
  var ltv         = inputs.customerLTV         || 5000;
  var insPremium  = inputs.insurancePremium    || 75000;
  var ipVal       = inputs.ipValue             || 2000000;

  // Look up tables
  var sectorData = SECTOR_DATA[sector] || SECTOR_DATA.technology;
```

```
var sizeMult   = SIZE_MULT[orgSize]  || 1.0;
var regData    = REG_DATA[regulation] || REG_DATA.none;
var currData   = CURRENCY[currency]  || CURRENCY.USD;
var sym        = currData.symbol;
var fx         = currData.rate;


// --- STEP 1: Base breach cost ---------------------------------------
//
// Start with the sector average for a large org, then scale to size.

var baseCost = sectorData.baseCost * sizeMult;

// --- STEP 2: Record volume adjustment -------------------------------
//
// More records at risk = higher breach cost (notification, fines).
// Formula: add up to 50% more cost, scaling linearly up to 100,000
// records. Hard cap at 1.5× so it doesn't run away.

var recordAdj = Math.min(1.5,  1 + (records / 100000) * 0.3);
baseCost = baseCost * recordAdj;

// --- STEP 3: Breach probability -------------------------------------
//
// Start with the sector's annual breach probability.
// Each prior incident adds 5% (attackers re-target known victims).
// Hard cap at 80% — even the most-breached orgs don't hit 100%.

var breachProb = Math.min(0.80,  sectorData.breachProb + (incidents * 0.05));

// --- STEP 4: Vendor risk modifier -----------------------------------
//
// Each high-access supplier adds 2% to the ALE.
// Capped at +40% (at 20+ suppliers). WEF 2026: 65% of large
// enterprises rank supply chain as their #1 resilience risk.

var vendorMod = 1 + Math.min(0.40,  vendors * 0.02);

// --- STEP 5: Regulatory fine ----------------------------------------
//
// Use the LOWER of: the framework's max fine, or the % of revenue.
// This prevents over-stating fines for very large enterprises.

var regFine = Math.min(regData.maxFine,  revenue * regData.pctRevenue);

// --- STEP 6: Downtime cost ------------------------------------------
//
```

```
// Simple: hourly cost × expected hours of outage.
// Default: $25,000/hr × 72 hrs = $1,800,000.

var downtimeCost = dtCostPerHr * dtHours;

// --- STEP 7: Reputational damage (customer churn) ------------------
//
// Estimate customer count from revenue and LTV.
// Apply churn rate, but only attribute 30% directly to the breach
// (not all post-breach churn is caused by the breach itself).

var estCustomers = revenue / (ltv * 10);
var churnLoss    = estCustomers * churnRate * ltv * 0.30;

// --- STEP 8: IP / proprietary data loss ----------------------------
//
// Assume 15% of IP value is exposed in a breach scenario.
// This is a conservative estimate — actual exposure varies widely.

var ipLoss = ipVal * 0.15;

// --- STEP 9: Threat multiplier -------------------------------------
//
// Sum the aleFactor values for all active threats.
// This creates a multiplier: 1.0 = average threat environment.
// Cap at 2.0 (maximum threat environment).
// Floor at 0.2 — residual risk always exists even with no threats selected.

var threatFactor = 0;
THREATS.forEach(function(threat) {
  if (activeThreats.indexOf(threat.id) !== -1) {
    threatFactor += threat.aleFactor;
  }
});
var normalizedThreat = Math.min(2.0,  Math.max(0.2, threatFactor));

// --- STEP 10: Raw ALE (Annual Loss Expectancy, pre-controls) -------
//
// The expected annual loss with no security investment.
// This is the number that gets reduced by your controls.
//
// Formula:
//   rawALE = (sum of all loss components)
//            × breachProbability
//            × vendorRiskModifier
//            × threatMultiplier
```

```
var rawALE = (baseCost + regFine + downtimeCost + churnLoss + ipLoss)
             * breachProb
             * vendorMod
             * normalizedThreat;

// --- STEP 11: Control risk reduction factor ------------------------
//
// Sum the riskReduction values for all active controls.
// Hard cap at 0.88 (88%) — no combination of controls eliminates
// all risk. Residual risk always remains.

var totalReduction = 0;
CONTROLS.forEach(function(control) {
  if (activeControls.indexOf(control.id) !== -1) {
    totalReduction += control.riskReduction;
  }
});
var reductionFactor = Math.min(0.88,  totalReduction);

// --- STEP 12: Mitigated ALE and annual savings ---------------------
//
// mitigatedALE = what you expect to lose WITH your controls.
// annualSavings = the difference — this is the value your security
//                 investment delivers each year.

var mitigatedALE  = rawALE * (1 - reductionFactor);
var annualSavings = rawALE - mitigatedALE;

// --- STEP 13: Return on Investment (ROI) ---------------------------
//
// Classic "loss avoided" ROI formula.
// Answers: for every $1 spent on security, how much loss is avoided?
//
// +150% means: for every $1 spent, $2.50 of loss is avoided
//  (after deducting the cost of the investment itself).
// Negative ROI means the budget exceeds the loss it prevents.

var roi = secBudget > 0
  ? ((annualSavings - secBudget) / secBudget) * 100
  : 0;

// --- STEP 14: Confidence intervals ---------------------------------
//
// Simple ±40% range around the expected ALE.
// These are deterministic multipliers — not statistical distributions.
// They communicate a realistic range, not false precision.
```

```
var aleWorstCase = rawALE * 1.4;
var aleExpected  = rawALE * 1.0;
var aleBestCase  = rawALE * 0.6;

// --- STEP 15: Payback period ----------------------------------------
//
// How many months until the security investment pays for itself?
// Minimum 1 month — avoids displaying 0 for very efficient programmes.

var paybackMonths = annualSavings > 0
  ? Math.max(1,  Math.round((secBudget / annualSavings) * 12))
  : 999;

// --- STEP 16: 5-Year Discounted Cash Flow ---------------------------
//
// Projects cumulative value of security investment over 5 years.
// threatGrowthRate = 7%  (conservative; historical is 15%+ CAGR)
// discountRate     = 8%  (mid-range corporate cost of capital)
// Starts negative (year 0 = -budget, the upfront investment cost)

var threatGrowthRate = 0.07;
var discountRate     = 0.08;
var dcfRows          = [];
var cumulative       = -secBudget;

for (var year = 1; year <= 5; year++) {
  var savingsGrowth   = annualSavings * Math.pow(1 + threatGrowthRate, year - 1
  var discountedValue = savingsGrowth / Math.pow(1 + discountRate, year);
  var netBenefit      = discountedValue - secBudget;
  cumulative         += discountedValue;

  dcfRows.push({
    year:           year,
    savingsGrowth:  savingsGrowth,
    discountedValue:discountedValue,
    netBenefit:     netBenefit,
    cumulative:     cumulative
  });
}

var fiveYearNPV = dcfRows.reduce(function(sum, row) {
  return sum + row.netBenefit;
}, 0);

// --- STEP 17: Insurance premium reduction ---------------------------
//
// Improved security posture can reduce cyber insurance premiums.
```

```
// Model: controls reduce premium by up to 25% proportionally.
// Only shown when saving > $500 (avoids trivially small amounts).

var insuranceSaving = insPremium * reductionFactor * 0.25;

// --- STEP 18: Security Health Score (300–850) ----------------------
//
// A single composite number for non-technical stakeholders.
// Ranges mirror a consumer credit score — immediately intuitive.
//
// Components:
//   Base score:     300 pts  (always)
//   controlScore:   0–300    (based on reductionFactor)
//   rosiScore:      0–100    (based on ROI %)
//   budgetScore:    0–100    (security budget as % of revenue)
//   paybackScore:   0–50     (based on payback period)
//   Maximum:        850

var controlScore  = reductionFactor * 300;
var rosiScore     = Math.min(100,  Math.max(0,  roi / 4));
var budgetScore   = Math.min(100,  (secBudget / revenue) * 1000);
var paybackScore  = paybackMonths <= 12  ? 50
                  : paybackMonths <= 24  ? 30
                  : paybackMonths <= 60  ? 15
                  : 0;

var healthScore = Math.round(
  Math.min(850,  300 + controlScore + rosiScore + budgetScore + paybackScore)
);

// Score band label
var scoreBand;
if      (healthScore >= 780) scoreBand = 'Strong programme';
else if (healthScore >= 700) scoreBand = 'Above average';
else if (healthScore >= 600) scoreBand = 'Getting there';
else if (healthScore >= 500) scoreBand = 'Gaps to close';
else                         scoreBand = 'Needs attention';

// --- STEP 19: Threat thermometer -----------------------------------
//
// Visual display only. Fills from left (safe) to right (exposed).
// 0% = fully protected (all controls active, reductionFactor = 0.88)
// 100% = no controls active (reductionFactor = 0)

var thermoValue = Math.round((1 - reductionFactor) * 100);
```

```
// --- APPLY CURRENCY ------------------------------------------------
//
// All monetary outputs are multiplied by the exchange rate.
// fx = 1.0 for USD, 0.79 for GBP, etc.

// --- RETURN ALL RESULTS --------------------------------------------

return {

  // Core financials (in selected currency)
  rawALE:           rawALE          * fx,
  mitigatedALE:     mitigatedALE    * fx,
  annualSavings:    annualSavings   * fx,
  roi:              roi,                    // percentage, no fx needed
  insuranceSaving:  insuranceSaving * fx,

  // Confidence intervals (in selected currency)
  aleWorstCase: aleWorstCase * fx,
  aleExpected:  aleExpected  * fx,
  aleBestCase:  aleBestCase  * fx,

  // Payback
  paybackMonths: paybackMonths,

  // 5-year DCF (all values in selected currency)
  dcfRows:     dcfRows.map(function(row) {
    return {
      year:            row.year,
      savingsGrowth:   row.savingsGrowth   * fx,
      discountedValue: row.discountedValue * fx,
      netBenefit:      row.netBenefit      * fx,
      cumulative:      row.cumulative      * fx
    };
  }),
  fiveYearNPV: fiveYearNPV * fx,

  // Security Health Score
  healthScore:  healthScore,
  scoreBand:    scoreBand,

  // Threat thermometer (0-100)
  thermoValue: thermoValue,

  // Intermediate values — useful for debugging or displaying sub-totals
  debug: {
    baseCost:        baseCost,
    recordAdj:       recordAdj,
```

```
      breachProb:      breachProb,
      vendorMod:       vendorMod,
      regFine:         regFine,
      downtimeCost:    downtimeCost,
      churnLoss:       churnLoss,
      ipLoss:          ipLoss,
      normalizedThreat:normalizedThreat,
      reductionFactor: reductionFactor,
      controlScore:    controlScore,
      rosiScore:       rosiScore,
      budgetScore:     budgetScore,
      paybackScore:    paybackScore
    },

    // Currency info — useful for formatting outputs
    currency: {
      code:   currency,
      symbol: sym,
      rate:   fx
    }
  };
}


// ============================================================
// SECTION 3 — HELPER FUNCTIONS
// Small utilities your UI will need.
// ============================================================

/**
 * formatMoney(value, symbol)
 * Formats a number as a short currency string.
 * e.g. 4880000 → "$4.9M"   52000 → "$52K"
 */
function formatMoney(value, symbol) {
  symbol = symbol || '$';
  if (isNaN(value) || value === null) return symbol + '0';
  var abs = Math.abs(value);
  var prefix = value < 0 ? '-' : '';
  if (abs >= 1000000) return prefix + symbol + (abs / 1000000).toFixed(1) + 'M';
  if (abs >= 1000)    return prefix + symbol + Math.round(abs / 1000) + 'K';
  return prefix + symbol + Math.round(abs).toLocaleString();
}

/**
 * formatPayback(months)
 * Converts a month count to a readable string.
```

```
 * e.g. 9 → "9 months"   18 → "18 months"   30 → "2.5 years"
 */
function formatPayback(months) {
  if (months >= 999) return '>10 years';
  if (months >= 24)  return (months / 12).toFixed(1).replace('.0', '') + ' years'
  return months + ' month' + (months === 1 ? '' : 's');
}

/**
 * gaugeOffset(score)
 * Maps a health score (300–850) to an SVG stroke-dashoffset value.
 * The gauge arc has a total length of 251.3 (from the SVG path).
 * Returns a dashoffset value — lower = more of the arc is filled.
 *
 * Usage in SVG:
 *   <path stroke-dasharray="251.3" stroke-dashoffset="{{ gaugeOffset(score) }}"
 */
function gaugeOffset(score) {
  var pct = (score - 300) / 550;  // 0 to 1
  pct = Math.max(0, Math.min(1, pct));
  return 251.3 * (1 - pct);
}

/**
 * gaugeColor(score)
 * Returns the hex colour for the arc gauge based on the score.
 *   300–499 → red
 *   500–699 → amber
 *   700–850 → green
 */
function gaugeColor(score) {
  if (score >= 700) return '#10B981';  // green
  if (score >= 500) return '#F59E0B';  // amber
  return '#EF4444';                    // red
}


// ============================================================
// SECTION 4 — WORKED EXAMPLE
// Copy these inputs into calculate() and verify your outputs
// match the expected values. Use this to confirm the formulas
// are implemented correctly before connecting the UI.
// ============================================================

/**
 * TEST INPUTS
 * ------------------------------------------------------------
```

```
* sector:         'healthcare'
* orgSize:        'mid'
* revenue:        50,000,000
* secBudget:      500,000
* records:        50,000
* priorIncidents: 0
* regulation:     'hipaa'
* vendors:        12
* activeControls: ['mfa', 'edr', 'siem', 'training', 'backup']
* activeThreats:  ['ransomware', 'phishing', 'supply']
* currency:       'USD'
* (all expert fields at default values)
*
* STEP-BY-STEP EXPECTED OUTPUTS
* ------------------------------------------------------------
* baseCost (before recordAdj):
*   11,200,000 × 0.70 (mid) = 7,840,000
*
* recordAdj:
*   min(1.5,  1 + (50,000 / 100,000) × 0.3) = min(1.5, 1.15) = 1.15
*
* baseCost (after recordAdj):
*   7,840,000 × 1.15 = 9,016,000
*
* breachProb:
*   min(0.8,  0.32 + (0 × 0.05)) = 0.32
*
* vendorMod:
*   1 + min(0.40,  12 × 0.02) = 1 + min(0.40, 0.24) = 1.24
*
* regFine (HIPAA):
*   min(1,900,000,  50,000,000 × 0.01) = min(1,900,000, 500,000) = 500,000
*
* downtimeCost (defaults):
*   25,000 × 72 = 1,800,000
*
* churnLoss (defaults — ltv=5000, churn=15%):
*   estCustomers = 50,000,000 / (5,000 × 10) = 1,000
*   churnLoss    = 1,000 × 0.15 × 5,000 × 0.30 = 225,000
*
* ipLoss (default — ipValue=2,000,000):
*   2,000,000 × 0.15 = 300,000
*
* normalizedThreat (ransomware + phishing + supply):
*   min(2.0,  0.65 + 0.55 + 0.45) = min(2.0, 1.65) = 1.65
*
* rawALE:
```

```
 *   (9,016,000 + 500,000 + 1,800,000 + 225,000 + 300,000)
 *   × 0.32 × 1.24 × 1.65
 *   = 11,841,000 × 0.32 × 1.24 × 1.65
 *   = 11,841,000 × 0.65472
 *   ≈ 7,752,300
 *
 * reductionFactor (mfa+edr+siem+training+backup):
 *   min(0.88,  0.28+0.22+0.18+0.15+0.20)
 *   = min(0.88, 1.03) = 0.88
 *   NOTE: these 5 controls sum to 1.03, which exceeds the cap.
 *   The cap of 0.88 applies.
 *
 * mitigatedALE:
 *   7,752,300 × (1 − 0.88) = 7,752,300 × 0.12 ≈ 930,276
 *
 * annualSavings:
 *   7,752,300 − 930,276 ≈ 6,822,024
 *
 * ROI:
 *   ((6,822,024 − 500,000) / 500,000) × 100
 *   = (6,322,024 / 500,000) × 100
 *   ≈ +1,264%
 *
 * paybackMonths:
 *   max(1,  round((500,000 / 6,822,024) × 12))
 *   = max(1, round(0.0733 × 12))
 *   = max(1, round(0.879))
 *   = max(1, 1) = 1 month
 *
 * healthScore:
 *   controlScore = 0.88 × 300 = 264
 *   rosiScore    = min(100, 1264 / 4) = min(100, 316) = 100
 *   budgetScore  = min(100, (500,000 / 50,000,000) × 1000)
 *                = min(100, 0.01 × 1000) = min(100, 10) = 10
 *   paybackScore = 50  (≤12 months)
 *   healthScore  = min(850, 300 + 264 + 100 + 10 + 50) = min(850, 724) = 724
 *   scoreBand    = 'Above average'
 *
 * thermoValue:
 *   round((1 − 0.88) × 100) = round(12) = 12%
 *
 * insuranceSaving:
 *   75,000 × 0.88 × 0.25 = 16,500
 * ------------------------------------------------------------
 */

// To run the test:
```

```
// var testInputs = {
//   sector: 'healthcare', orgSize: 'mid', revenue: 50000000,
//   secBudget: 500000, records: 50000, priorIncidents: 0,
//   regulation: 'hipaa', vendors: 12,
//   activeControls: ['mfa','edr','siem','training','backup'],
//   activeThreats: ['ransomware','phishing','supply'],
//   currency: 'USD'
// };
// console.log(calculate(testInputs));
```