เผื่อไว้เท่าไรถึงพอ

สารานุกรมความปลอดภัย ตั้งแต่ค่าความปลอดภัยของวิศวกร ถึงศัตรูที่อ่านแบบแปลนของคุณ

ห้องสมุด AICE · how-much-margin-is-enough
8 บท · ฉบับภาษาเดียว
เขียนครั้งแรก 11 กันยายน 2026 · ปรับปรุงล่าสุด 11 กันยายน 2026
ทุกตัวเลข ชื่อ และปีในเล่มนี้ตรวจย้อนไปยังแหล่งที่อ้างไว้ได้

ก่อนอ่าน

วิศวกรที่ออกแบบสะพานรู้ว่าต้องเผื่อเท่าไร เขามีตัวเลข มีตาราง มีประวัติของสะพานที่พังมาก่อนหน้าให้เรียนรู้ และการเผื่อของเขาได้ผลจริง สะพานเกือบทุกแห่งในโลกยืนอยู่ได้เพราะวิธีคิดแบบนี้ ทีนี้เอาวิธีคิดเดียวกันไปใช้กับรหัสผ่าน กับเซิร์ฟเวอร์ กับ AI agent ที่อ่านอีเมลแทนคุณ แล้วมันจะพัง ไม่ใช่เพราะเผื่อน้อยไป แต่เพราะมันเป็นปัญหาคนละชนิด

ความต่างอยู่ตรงเดียว **แรงลมไม่เคยอ่านแบบก่อสร้างของคุณ** มันไม่รู้ว่าเสาต้นไหนบาง มันไม่รอให้ฝนตกก่อนค่อยพัด มันมาแบบเดิมทุกครั้ง ส่วนคนที่อยากเข้าระบบของคุณอ่าน เขาหาจุดที่บางที่สุดโดยเฉพาะ และถ้าคุณเสริมจุดนั้น เขาก็ย้ายไปจุดถัดไป ส่วนเผื่อจึงเป็นกันชนในโลกหนึ่ง และเป็นแค่ตัวเลขที่ถูกวัดแล้วเดินอ้อมในอีกโลกหนึ่ง

เล่มนี้คือห้าสิบสี่เรื่องที่พาไปตามเส้นนั้น จากค่าความปลอดภัยที่วิศวกรยุค 1850 คิดขึ้นมา ผ่านสะพานที่บิดตัวจนขาด เรือที่มีเรือชูชีพเกินกฎแต่ไม่พอสำหรับคนครึ่งลำ กุญแจที่ถูกสะเดาะต่อหน้าคนดูในปี 1851 รหัสที่ชื่อว่าถอดไม่ได้ หนอนตัวแรกที่วิ่งข้ามอินเทอร์เน็ต ไปจนถึงเครื่องมือที่คุณใช้อยู่ทุกวันนี้ซึ่งอ่านคำสั่งจากคนอื่นได้ แต่ละเรื่องจบในตัว มีวันที่ มีตัวเลข และมีชื่อของสิ่งที่พัง เปิดตรงไหนก็อ่านได้

สิ่งที่เล่มนี้ไม่ทำ

ไม่มีโค้ดโจมตี ไม่มีขั้นตอนที่เอาไปทำตามได้ ทุกกลไกในเล่มอธิบายระดับแนวคิดเท่านั้น เพราะจุดประสงค์คือให้เข้าใจว่าอะไรพังและพังเพราะอะไร ไม่ใช่ให้ใครทำซ้ำ

ตัวเลขที่เล่มนี้ปฏิเสธจะพิมพ์ และทำไม

เรื่องความมั่นคงปลอดภัยมีนิสัยแย่อย่างหนึ่ง คือตัวเลขความเสียหายถูกเล่าต่อกันจนไม่มีใครจำได้ว่าใครพูดก่อน พอไล่กลับไปหาต้นทางจริงๆ มักไม่มีอะไรอยู่ที่นั่น

ตัวเลขเจ็ดตัวนี้ค้นแล้วไม่พบแหล่งชั้นต้น — ความเสียหายจากหนอน Morris หนึ่งร้อยล้านดอลลาร์ · การโจมตี Dyn ที่ 1.2 เทราบิตต่อวินาที · Stuxnet ใช้ zero-day ห้าตัว · Log4j มีผู้ดูแลสามคน · SolarWinds ถูกเจาะหนึ่งหมื่นแปดพันองค์กร · ค่าไถ่ Colonial Pipeline ที่ได้คืน · และความเสียหาย NotPetya หนึ่งหมื่นล้านดอลลาร์ ซึ่งเล่มนี้พิมพ์ในฐานะคำกล่าวของทำเนียบขาว วางคู่กับตัวเลขที่บริษัทเปิดเผยเอง

ทั้งเจ็ดตัวยังอยู่ในเล่ม พร้อมคำกำกับว่าหาต้นทางไม่ได้ เพราะประโยคที่อธิบายว่าทำไมตัวเลขถึงหายไป มักน่าสนใจกว่าตัวเลขเอง

ข้อที่ยืนยันไม่ได้ และเล่มนี้บอกตรงๆ ว่ายืนยันไม่ได้

ใครเป็นคนแรกที่ใช้คำว่า factor of safety · Pugsley เป็นคนบัญญัติคำว่า factor of ignorance หรือไม่ · สัดส่วนระหว่างคณิตศาสตร์กับความผิดพลาดของคนใช้งาน ในการถอดรหัส Enigma · ความถี่ที่ใช้งานจริงของอุปกรณ์ดักฟังในตราประทับของทูตอเมริกัน · คำกล่าวอ้างเรื่องฟิชชิงหลังการแจกกุญแจฮาร์ดแวร์ ที่บริษัทหนึ่งเผยแพร่โดยไม่มีงานวิจัยรองรับ · และตัวเลขร้อยละที่คนอ้างว่ามาจากหน่วยงานความมั่นคงไซเบอร์ของอเมริกา แต่ไม่ปรากฏในเอกสารของหน่วยงานนั้นเลย

คำที่ให้เครดิตผิดกันมานาน

ข้อความปี 1853 ที่เถียงว่าการเปิดเผยช่องโหว่เป็นประโยชน์ต่อคนซื่อสัตย์ มักถูกยกให้ A. C. Hobbs เจ้าของมือที่สะเดาะกุญแจในปี 1851 แหล่งชี้ว่าผู้เรียบเรียงคือ Charles Tomlinson ทำงานจากเอกสารของ Hobbs ภาคที่ 3 พิมพ์เครดิตตามนั้น

ข้อที่มาจากผู้ขายฝ่ายเดียว

ภาคที่ 6 ว่าด้วยยุค AI agent เต็มไปด้วยคำกล่าวอ้างที่มีบริษัทผู้รายงานเป็นแหล่งเดียว และยังไม่มีฝ่ายอิสระยืนยัน ทุกข้อแบบนั้นมีคำกำกับไว้ในเนื้อหา พร้อมชื่อผู้รายงานและชื่อคนที่คัดค้านต่อสาธารณะ เล่มนี้ไม่ตัดสินข้อพิพาทเหล่านั้น แต่พิมพ์ไว้ทั้งสองด้าน เพราะในวันที่เขียน ยังไม่มีใครเปิดหลักฐานให้ใครตรวจ

ลิงก์ในเล่มนี้กดได้ และแหล่งอ้างอิงปิดท้ายภาคของตัวเอง ไม่ได้กองรวมไว้ท้ายเล่ม เพราะแหล่งที่ตามกลับไปหาไม่ได้ ไม่ใช่แหล่ง

1 · ส่วนเผื่อคืออะไร

วิศวกรที่ต้องเซ็นชื่อรับรองคานตัวหนึ่ง รู้ภาระที่คานจะต้องรับ รู้กำลังของวัสดุโดยประมาณ แล้วยังเหลือคำถามอีกข้อที่การคำนวณไม่ตอบให้ คือจะใส่ความแข็งแรงเกินไปเท่าไรจึงจะพอ

ส่วนเผื่อคือคำตอบของคำถามนั้น มันคือความแข็งแรงที่ใส่เกินจากภาระที่คาดไว้ และตัวเลขที่บอกว่าเกินเท่าไรไม่ได้ลอยมา มีคนตั้ง มีปีที่ตั้ง และมีเหตุผลที่ตรวจได้ ภาคนี้ไล่ตัวเลขนั้นกลับไปหาต้นทาง

สิ่งที่ทำให้วิธีนี้ทำงานได้มีเงื่อนไขข้อเดียว ทุกกรณีในภาคนี้อยู่ในโลกที่สิ่งที่มาทำลายไม่ได้อ่านแบบ — ลม น้ำหนัก เชือกที่ล้า ไฟ แผ่นดินไหว ลมไม่รู้ว่าเสาต้นไหนบางที่สุด มันพัดใส่ทั้งอาคารเท่ากันหมด และเพราะมันไม่รู้ การใส่เผื่อเข้าไปอีกหน่อยจึงได้ผล

ค่าความปลอดภัย · 1858–1940s

ในปี 1858 W. J. M. Rankine พิมพ์ A Manual of Applied Mechanics ฉบับแรก ตำราที่วางสัดส่วนความปลอดภัยบนฐานของความเค้นให้วิศวกรอังกฤษ ปัญหาที่ตำราเล่มนั้นแก้ไม่ใช่การคำนวณ คนยุคนั้นคำนวณคานเป็น ปัญหาคือไม่มีใครรู้ว่าเหล็กหล่อก้อนตรงหน้าแข็งแรงเท่าค่าเฉลี่ยในตารางหรือเปล่า

คานเหล็กหล่อในยุคนั้นจึงถูกออกแบบด้วย factor of safety เท่ากับ 4 บนค่า ultimate tensile strength เฉลี่ย 93 N/mm² จนได้ความเค้นดัดที่ยอมให้ 23 N/mm² เลข 4 ไม่ได้มาจากสมการไหน มันคือที่ว่างที่วิศวกรกันไว้ให้สิ่งที่เขายังไม่รู้

Hansson แจงว่าตัวคูณตัวเดียวนี้รับอะไรไว้บ้าง มีห้าอย่าง: ภาระสูงกว่าที่คาด วัสดุด้อยกว่าที่คาด ทฤษฎีของกลไกวิบัติที่ไม่สมบูรณ์ กลไกวิบัติที่ยังไม่รู้จัก และคนคำนวณหรือออกแบบผิด สองข้อแรกคือความแปรปรวนของโลก สามข้อหลังคือความไม่รู้ของวิศวกร

ทั้งห้าข้อมาจากฝั่งเดียวกันหมด ไม่มีข้อไหนเป็นภาระที่เลือกลงตรงจุดที่บางที่สุด นั่นคือเหตุผลที่เลขตัวเดียวพอ — ไม่มีใครยืนอยู่อีกฝั่งคอยดูว่ามันถูกตั้งไว้เท่าไร

ต้นกำเนิดของตัวเลขนี้ แหล่งสองแหล่งไม่ตรงกัน และภาคนี้พิมพ์ทั้งสอง Isaac Elishakoff ย้อนแนวคิดไปถึงปี 1729 ในตำราป้อมปราการของ Bernard Forest de Bélidor ส่วน Sven Ove Hansson ระบุว่าการใช้ factor of safety อย่างชัดแจ้งครั้งแรกสุดที่รู้กันอยู่ในทศวรรษ 1860 ข้ออ้างที่ว่า Rankine เป็นผู้ใช้คำนี้คนแรกตามบันทึกของ OED ตรวจไม่ได้ในรอบค้นคว้านี้

ชื่อที่ตรงกว่าสำหรับตัวคูณนี้มีคนเสนอไว้แล้ว Alasdair Beal ตั้งชื่อบทความปี 2001 ใน The Structural Engineer ว่า "Factors of Ignorance?" ส่วนคำกล่าวที่ว่า Alfred Pugsley เป็นผู้บัญญัติวลีนี้ ไม่มีแหล่งใดยืนยัน สิ่งที่ Pugsley ทำจริงคือบทความปี 1951 ใน Journal of the Institution of Civil Engineers และเหรียญทองของ Institution of Structural Engineers ปี 1968 สำหรับปรัชญาความปลอดภัยที่มองโครงสร้างเป็นความน่าจะเป็นของการวิบัติ

ตัวเลขของเหล็กในอังกฤษเป็นซากสงคราม BS 449 ปี 1932 ใช้ตัวคูณราว 2 สำหรับคานเหล็ก ฉบับแก้ไขเดือนพฤศจิกายน 1939 ลดเหลือ 1.6 "as an emergency measure" พร้อมข้อความว่าให้สิ้นผลหกเดือนหลังสงครามยุติ คำว่า emergency measure บอกอายุของตัวเองไว้ในตัว แล้วปี 1948 มันถูกทำให้ถาวรที่ 1.65–1.7 และอยู่อย่างนั้นตลอดยุค permissible-stress design

เครื่องบินเดินคนละทางแต่ลงเอยแบบเดียวกัน ตัวเลข 1.5 ถูกตั้งขึ้นทีละส่วน ปี 1930 US Air Corps ใช้ 1.5 กับภาระออกแบบของหางเครื่องบินก่อน เดือนมีนาคม 1934 Revision G ของ Handbook of Instructions for Airplane Design ทำให้มันเป็นข้อกำหนดทั่วไป และปัจจุบันอยู่ใน 14 CFR 25.303 ผู้เขียนประวัติของตัวเลขนี้ที่ NASA สรุปเองว่ามันไม่มีต้นกำเนิดเดียว "much like the rules of baseball"

ตัวเลข 1.5 ตั้งใจให้ครอบคลุมสามอย่าง: ภาระใช้งานที่เกินขีดออกแบบ ตัวโครงสร้างโก่งเกินขีดจำกัด และชิ้นส่วนที่บางกว่าที่คำนวณแต่ยังอยู่ในพิกัดยอมรับ มันไม่ครอบคลุมความผิดพลาดในการวิเคราะห์โครงสร้าง และราคาของมันวัดได้ ลดจาก 1.5 เป็น 1.4 ประหยัดน้ำหนักโครงสร้างได้ 4%

ตัวเลขฉุกเฉินที่เขียนวันหมดอายุไว้ในตัวเอง กลายเป็นตัวเลขถาวร ตัวเลขของเครื่องบินที่ไม่มีต้นกำเนิดเดียว กลายเป็นกฎหมาย ทั้งสองตัวอยู่ได้เพราะสิ่งที่มันกันอยู่ไม่เคยเปลี่ยนวิธีเล่น

สิ่งที่กรณีนี้สอน: ก่อนเชื่อตัวคูณใด ให้เขียนรายการห้าข้อของ Hansson ไว้ข้างมัน แล้วขีดว่าตัวคูณนั้นถูกตั้งขึ้นเพื่อรับข้อไหน — ข้อที่ไม่ได้ขีดคือส่วนที่ไม่ได้เผื่อ

เบรกลิฟต์ของโอทิส · 1853

ปัญหาของแท่นยกที่ห้อยอยู่กับเชือกมีข้อเดียว คือเชือกขาด และทางแก้ที่ตรงที่สุดคือทำเชือกให้หนาขึ้น เชือกที่หนาขึ้นกันได้เฉพาะเหตุที่คิดออกแล้วเท่านั้น

สิ่งที่ Elisha Otis ทำไม่ใช่อย่างนั้น เขาปล่อยให้เชือกขาดต่อไปตามเดิม แล้วไปจัดการกับอีกด้านหนึ่งแทน คือทำให้การขาดของเชือกไม่สำคัญ

วันศุกร์ที่ 5 พฤษภาคม 1854 สมุดบัญชีของ Otis บันทึกรับเงิน 100 ดอลลาร์ค่า "one model hoist machine" ส่งไปยัง New York Crystal Palace อาคารที่สร้างขึ้นในปี 1853 เพื่อเลียนแบบงานแสดงใหญ่ของลอนดอน

อาคารนั้นเปิดวันที่ 4 กรกฎาคม 1853 และขาดทุน ต้นปี 1854 สมาคมเจ้าของจัดโครงสร้างใหม่ ตั้ง P. T. Barnum เป็นประธาน ปิดอาคารตั้งแต่ 15 เมษายน แล้วเปิดใหม่วันที่ 4 พฤษภาคม ด้วยแผนกเครื่องจักรที่ขยายใหญ่ขึ้นและเดินด้วยไอน้ำ วันที่ 5 พฤษภาคม Barnum ประกาศรางวัลเหรียญทองมูลค่า 1,000 ดอลลาร์สำหรับสิ่งประดิษฐ์ที่มีประโยชน์ที่สุดแห่งปี และเหรียญละ 100 ดอลลาร์อีกห้าเหรียญ

บันทึกร่วมสมัยชิ้นแรกคือ New York Tribune วันที่ 30 พฤษภาคม 1854 มันเล่าว่านักประดิษฐ์ขึ้นลงบนแท่นยก และ "occasionally cuts the rope by which it is supported"

กลไกที่เขาตัดเชือกให้ดูนั้นเรียบง่ายจนน่าหงุดหงิด The Practical Mechanic's Journal ของอังกฤษ ฉบับกรกฎาคม 1854 อธิบายมันและพิมพ์ภาพแรก เสาไม้ตั้งมีฟันเฟืองเรียงเป็นแถวเหมือนฟันเลื่อย แผ่นเบรกถูกสปริงแข็งดันให้อยากเกาะฟันนั้น

อยู่ตลอดเวลา และสิ่งเดียวที่รั้งมันไว้คือแรงตึงของเชือกที่ดึงขึ้นไป พอเชือกขาด แรงตึงหายไปพร้อมกัน สปริงจึงได้ทำในสิ่งที่มันอยากทำมาตลอด

บทความเดียวกันถูกพิมพ์ซ้ำใน Dinglers Polytechnisches Journal ของเยอรมนี เดือนธันวาคม 1854 พร้อมข้อสังเกตว่าอุปกรณ์แบบเดียวกันมีใช้ในปล่องลิฟต์มานานแล้ว Charles Otis ลูกชาย บันทึกในปี 1911 ว่าแท่นนั้นเคลื่อนขึ้นลงได้ราว 30–40 ฟุต

ตำนานที่เล่ากันตายไปหลายเรื่องในรอบค้นคว้านี้ สองเรื่องที่หนักที่สุดเป็นแบบนี้ เรื่องแรก มันไม่ใช่การแสดงครั้งเดียวในบ่ายวันหนึ่ง คำว่า occasionally ในข่าวของ Tribune บอกเองว่าเป็นนิทรรศการต่อเนื่องหลายสัปดาห์ ตัดเชือกซ้ำแล้วซ้ำอีก จึงไม่มีวันที่เดียวให้อ้าง เรื่องที่สอง ประโยค "All safe, gentlemen, all safe" ไม่มีแหล่งร่วมสมัยไหนบันทึก Lee Gray ผู้ค้นเอกสารเหล่านี้สรุปว่าเรื่องเล่าฉบับที่รับกันมา "relies more on faith and trust in the storytellers than on actual evidence"

สิ่งที่ยืนยันได้คือคำสั่งซื้อเริ่มเข้ามาต้นเดือนมิถุนายน 1854 และลิฟต์โดยสารเชิงพาณิชย์ตัวแรกเปิดใช้ในนิวยอร์กวันที่ 23 มีนาคม 1857 ข้ออ้างว่าการสาธิตนี้ทำให้ตึกสูงเป็นไปได้ เป็นการตีความของนักประวัติศาสตร์ ไม่ใช่ข้อเท็จจริงที่มีแหล่ง

เบรกตัวนี้ไม่เคยถามว่าเชือกขาดเพราะอะไร มันรออยู่อย่างเดียวคือแรงตึงที่หายไป นั่นแปลว่ามันกันเหตุที่ Otis เองยังคิดไม่ถึงได้ด้วย และเชือกที่ล้าไม่เคยเลือกเวลาขาดให้ตรงกับตอนที่สปริงติดขัด

สิ่งที่กรณีนี้สอน: ตัดเชือกเองต่อหน้าคนดู และตัดซ้ำ — อุปกรณ์นิรภัยที่ยังไม่เคยถูกสั่งให้ทำงานจริง ยังไม่ใช่อุปกรณ์นิรภัย

จากตัวคูณเดียว สู่ตัวคูณแยกส่วน · LRFD

ตัวคูณตัวเดียวใช้ได้ดีจนถึงวันที่มีคนอยากแก้มัน ในเลขตัวนั้นไม่มีใครชี้ได้ว่าส่วนไหนเผื่อไว้ให้ลม ส่วนไหนเผื่อไว้ให้คอนกรีตที่เทไม่ได้เรื่อง และเมื่อแยกไม่ออก ก็ปรับทีละส่วนไม่ได้ ต้องขยับทั้งก้อนหรือไม่ขยับเลย

ในปี 1986 AISC พิมพ์ Load and Resistance Factor Design Specification for Structural Steel Buildings ฉบับแรก หกสิบสามปีหลังสเปกฉบับแรกของสถาบันเดียวกันในปี 1923 ที่ใช้ตัวคูณตัวเดียวหารความเค้นวิบัติ

วิธีเดิมชื่อ Allowable Stress Design เอาความเค้นที่วัสดุรับได้หารด้วยตัวเลขเดียว แล้วเทียบความเค้นที่คำนวณได้กับผลหารนั้น ภาระต่างชนิดถูกจัดการอย่างหยาบ เช่นชิ้นส่วนสะพานตรวจภาระตายรวมภาระจรที่ 100% ของค่ายอมให้ และภาระตายรวมลมที่ 125%

LRFD เปลี่ยนสองอย่าง ภาระแต่ละชนิดได้ตัวคูณของตัวเอง ภาระตายต่ำ ภาระจร ลม หิมะ สูง เพราะแต่ละอย่างแปรปรวนไม่เท่ากัน และมี resistance factor ลดกำลังที่คำนวณได้ลง เพื่อรับความแปรปรวนของวัสดุ ของสมมติฐานที่ใช้คำนวณ ของงานประกอบและงานติดตั้ง จุดเทียบย้ายจากความเค้นไปเป็นกำลัง

สิ่งที่เปลี่ยนคือวิธีทำบัญชี ไม่ใช่ยอดในบัญชี ตัวคูณย่อยชุดแรกไม่ได้ถูกคำนวณขึ้นใหม่จากสถิติการวิบัติ มันถูกหารออกมาจากเลขตัวเดิม แล้วขยับให้ผลไม่เท่าเดิมพอดี

คอนกรีตไปก่อน แรงผลักคือวัสดุขาดแคลนในยุโรปหลังสงคราม ACI ปี 1963 ผลักงานออกแบบคอนกรีตจาก Working Stress ไป Ultimate Strength และปี 1971 ลดวิธีเดิมเป็น "alternate method" AISC ออก ASD ฉบับสุดท้ายปี 1989 และรวมสองวิธีไว้ในเล่มเดียวปี 2005 ฝั่งสะพาน AASHTO ผ่านขั้นกลางชื่อ Load Factor Design ปี 1973 พิมพ์ LRFD ฉบับแรกปี 1994 และหยุดปรับปรุงฉบับ ASD หลังปี 2002

ที่มาของตัวคูณย่อยอยู่ในบันทึกที่วิชาชีพไม่ได้โฆษณา Beal บันทึกจาก W. B. Cranston ว่าคณะกรรมการ CP110 ของอังกฤษปี 1972 อยากได้ตัวคูณที่ derive จากสถิติ แต่ข้อมูลไม่มี จึงเอา 1.8 ของ CP114 มาแยกส่วน ตัวคูณด้านภาระ 1.5 คูณด้วยตัวคูณด้านวัสดุ 1.2 ได้ 1.8 พอดี คือเท่าโค้ดเดิมทุกประการ และนั่นคือสิ่งที่คณะกรรมการไม่เอา "but the complexity of the new code would then have seemed pointless" จึงตั้ง 1.6 สำหรับภาระจร 1.4 สำหรับภาระตาย และ 1.15 สำหรับวัสดุ คำนำของ CP110 เองรับว่าข้อมูลสถิติที่เกี่ยวข้องมีไม่พอ และค่าที่ใช้ "based on current practice"

ฝั่งเหล็กเดินซ้ำรอยเดียวกัน ร่างโค้ดปี 1981 ตั้งตัวคูณด้านวัสดุที่ 1.075 เพื่อให้ผลใกล้ BS 449 วิศวกรปฏิเสธโค้ดที่ซับซ้อนขึ้นแต่ไม่ประหยัดขึ้น BS 5950 จึงลดตัวคูณด้านวัสดุเหลือ 1.0 เมื่อ BSI ถอน BS 8110 ในปี 2010 มันยังใช้ตัวคูณย่อยที่ตั้งไว้ชั่วคราวเมื่อปี 1972 Eurocode 0 ปี 2002 หมวด C4 ระบุว่าวิธีความน่าจะเป็นเต็มรูปแบบ "are seldom used" เพราะขาดข้อมูล และฐานทั่วไปของตัวคูณย่อยคือ "calibration to a long experience of building tradition"

บัญชีที่ละเอียดขึ้นไม่ใช่ของเปล่าประโยชน์ ความแปรปรวนของภาระถูกเขียนลงชัดและแก้ได้ทีละตัวแล้ว แต่บัญชีแบบนี้ทำได้เพราะภาระไม่ตอบสนองต่อวิธีที่มันถูกลงบัญชี ลมไม่แรงขึ้นเพราะตัวคูณของมันถูกตั้งไว้สูง

สิ่งที่กรณีนี้สอน: เจอตัวคูณย่อยชุดไหน ให้คูณกลับเป็นเลขเดียวแล้วเทียบกับโค้ดรุ่นก่อน — ถ้าเท่ากัน ตัวเลขนั้นถูกแบ่ง ไม่ได้ถูก derive

fail-safe กับ fail-secure

คนที่ลงบันไดหนีไฟแล้วพบว่าลงต่อไปไม่ได้ เหลือทางเดียวคือกลับเข้าชั้น ประตูบานนั้นต้องยอมให้เขาเข้า และประตูบานเดียวกันนั้นเองเป็นประตูกันไฟ หน้าที่ของมันคือกลอนให้แน่นไม่ให้ไฟลามเข้าปล่องบันได

International Building Code ฉบับปี 2024 สั่งให้ล็อกไฟฟ้าบนประตูบันไดหนีไฟปลดเมื่อไฟดับ ส่วน NFPA 80 สั่งให้ electric strike บนประตูกันไฟล็อกเมื่อไฟดับ — ประตูบานเดียว ข้อบังคับสองข้อ ทิศตรงข้าม

นิยามตาม Lori Greene แห่ง Allegion: fail safe คือปลดล็อกฝั่งเข้าเมื่อไฟดับ ต้องจ่ายไฟเพื่อล็อก fail secure คือล็อกฝั่งเข้าเมื่อไฟดับ ต้องจ่ายไฟเพื่อปลด ทั้งสองคำพูดถึงฝั่งที่มีกุญแจ ไม่ใช่ฝั่งคนอยู่ข้างใน ผลิตภัณฑ์ส่วนใหญ่ให้ออกได้เสรีทั้งสองแบบ คนข้างในบิดมือจับหรือดันคานแล้วออกได้เสมอ

ชื่อทั้งสองตั้งจากมุมของไฟฟ้า ไม่ใช่มุมของคนที่ยืนอยู่หน้าประตู และคำถามที่ชื่อทั้งสองไม่ถามเลยคือ ไฟดับเพราะอะไร

ประตูบันไดหนีไฟคือจุดที่ข้อบังคับชนกันจริง ล็อกฝั่งบันไดต้องเป็น fail safe เพราะคนที่กลับเข้าชั้นไม่ได้คือคนที่ติดอยู่ในปล่อง แต่ NFPA 80 กำหนดให้ electric strike ที่ติดบนชุดประตูกันไฟเป็น fail secure เพราะประตูต้องกลอนแน่นเมื่อไฟไหม้ ทางออกคือไม่ใช้ electric strike ใช้ล็อกไฟฟ้าเชิงกลแบบ fail safe, trim แบบ fail safe บน fire exit hardware หรือแม่เหล็กไฟฟ้า

Greene แยกให้ชัดว่า electric lock บนประตูกันไฟไม่ถูกบังคับให้ fail secure มีแต่ electric strike เท่านั้น เธอเขียนถึงด้านตรงข้ามของความผิดพลาดว่า electric strike แบบ fail secure ที่ขังคนไว้ในปล่องบันได "is one of the most dangerous code violations a building can have"

ตัวฮาร์ดแวร์ปิดทางเลือกไปแล้วครึ่งหนึ่ง แม่เหล็กไฟฟ้ามีแต่แบบ fail safe เพราะไม่มีไฟก็ไม่มีแรงยึด และเพราะมันไม่ให้ทางออกเสรีในตัว โค้ดจึงบังคับอุปกรณ์ปลดแยกต่างหาก ทั้งปุ่มข้างประตู สัญญาณเพลิงไหม้ และไฟดับ ส่วน electric latch retraction มีแต่แบบ fail secure กลอนจะยื่นออกเมื่อไฟหาย

IBC 2024 กำหนดให้ล็อกบันไดแบบ fail safe ปลดเมื่อได้สัญญาณจากศูนย์บัญชาการดับเพลิง เมื่อสัญญาณเพลิงไหม้ทำงานในพื้นที่ที่บันไดนั้นรับใช้ และเมื่อไฟฟ้าของระบบล็อกดับ delayed egress คือการซื้อเวลาต่อความขัดแย้งนี้ตรงๆ แหล่งรองระบุว่าประตูต้องเปิดภายใน 15 วินาทีหลังคนพยายามออก ตัวเลขนี้ตรวจกับตัวบท NFPA 101 ไม่ได้ในรอบค้นคว้านี้

Greene ปิดท้ายด้วยข้อเตือนที่พลิกด้าน: ผลิตภัณฑ์ fail safe จะปลดล็อกทุกครั้งที่มีสัญญาณเพลิงไหม้หรือไฟดับ "which is an obvious security risk" อ่านรายการเงื่อนไขปลดล็อกอีกครั้งแล้วจะเห็นว่าทุกข้อเป็นเงื่อนไขที่สร้างขึ้นได้ ไฟไหม้ไม่เลือกเวลา ส่วนสวิตช์มีคนกด

สิ่งที่กรณีนี้สอน: ก่อนเลือกล็อกไฟฟ้าทุกตัว เขียนลงกระดาษว่าเมื่อไฟดับมันอยู่ในสถานะไหน และใครบ้างที่ทำให้ไฟดับได้ — แล้วอ่านสองคำตอบนั้นเทียบกับข้อบังคับทุกข้อที่ประตูบานนั้นอยู่ใต้

การป้องกันหลายชั้น · INSAG-10, 1996

ป้อมที่มีคูน้ำ กำแพงนอก กำแพงใน และหอคอย ไม่ได้ถูกสร้างขึ้นเพื่อกันข้าศึกให้ขาด มันถูกสร้างเพื่อถ่วงเวลา ทุกชั้นที่ข้าศึกต้องผ่านคือเวลาที่ฝ่ายในได้เพิ่ม และสมมติฐานของป้อมมีข้อเดียว คือข้าศึกต้องผ่านทีละชั้น

ในปี 1996 International Nuclear Safety Advisory Group ของ IAEA พิมพ์ INSAG-10 Defence in Depth in Nuclear Safety และวางการป้องกันของโรงไฟฟ้านิวเคลียร์ไว้ห้าระดับในตารางเดียว ชื่อถูกยืมมาจากป้อม และสมมติฐานถูกยืมมาด้วย

คำนี้เป็นศัพท์ทหาร Thomas Wellock นักประวัติศาสตร์ของ NRC สืบว่ากองทัพฝรั่งเศสและเยอรมันในสงครามโลกครั้งที่หนึ่งคิดยุทธวิธีชื่อ defense in depth เพื่อกันข้าศึกทะลุแนวหน้า ด้วยระบบสนามเพลาะและจุดแข็งที่ซ้อนกันลึกหลายชั้น

เข้าสู่วิชานิวเคลียร์ทางโครงการแมนฮัตตัน ผู้รับผิดชอบเตาปฏิกรณ์ผลิตพลูโทเนียมที่ Hanford กลัวหายนะที่ปล่อยรังสีด้วยความรุนแรงระดับระเบิด จึงวาง "lines of defense" หลายแนว: ผู้ควบคุมที่ฝึกมา แท่งเชื้อเพลิงที่ผนึก กำแพงกำบัง ระบบหล่อเย็นและไฟฟ้าสำรอง และตัวสำรองของระบบดับเตาสำรอง ที่ถึงขั้นทำลายเตาเพื่อรักษาชีวิตผู้ควบคุม เรียกด้วยศัพท์ทหารอีกคำว่า last ditch

หลังสงคราม คณะกรรมการความปลอดภัยของ AEC จัดแนวเหล่านี้เป็นสามหน้าที่ คือ คุณสมบัติที่ทำให้เตาปลอดภัยในตัว สิ่งกีดขวางทางกายภาพเช่นอาคารคลุม และระบบเชิงรุกเพื่อดับและหล่อเย็น วลี defense in depth ปรากฏครั้งแรกราวปี 1958 ในงานสกัดพลูโทเนียมที่ Hanford และปี 1965 Glenn Seaborg ประธาน AEC ใช้มันในจดหมายถึงสภาคองเกรสสำหรับเตาปฏิกรณ์พลเรือน

INSAG-10 จัดห้าระดับตามลำดับ: หนึ่ง กันไม่ให้เครื่องเดินผิดปกติหรือขัดข้อง ด้วยแบบที่เผื่อไว้และงานก่อสร้างคุณภาพสูง สอง ควบคุมสภาพผิดปกติและตรวจจับความขัดข้อง สาม ควบคุมอุบัติเหตุภายในฐานออกแบบ สี่ ควบคุมสภาพรุนแรงและบรรเทาผลของอุบัติเหตุรุนแรง ห้า บรรเทาผลทางรังสีเมื่อสารกัมมันตรังสีรั่วออกมาก ด้วยแผนฉุกเฉินนอกโรงไฟฟ้า ย่อหน้า 21 ระบุเงื่อนไขที่ทุกระดับต้องมีก่อน คือ appropriate conservatism, quality assurance และ safety culture

สองย่อหน้าที่รับน้ำหนักทั้งรายงานคือ 22 กับ 23 ย่อหน้า 22: ความขัดข้องเดี่ยวที่ระดับใดระดับหนึ่ง หรือแม้ความขัดข้องหลายจุดในมากกว่าหนึ่งระดับ ต้องไม่ลามไปทำลายระดับถัดไป และ "the independence of different levels of defence is a key element" ย่อหน้า 23 ปิดข้ออ้างที่คนชอบใช้ที่สุด คือการมีหลายระดับ "does not justify continued operation in the absence of one element"

แล้วรายงานก็เขียนข้อยกเว้นของตัวเองไว้ที่ย่อหน้า 27 ไฟ น้ำท่วม แผ่นดินไหว อาจทำลายหลายระดับพร้อมกัน เพราะมันทั้งก่อสถานการณ์อุบัติเหตุ และทำลายเครื่องมือรับมือในคราวเดียว ห้าระดับเป็นอิสระต่อกันได้ ก็ต่อเมื่อสิ่งที่มากระทบไม่ได้เลือกกระทบทุกระดับพร้อมกัน

รายงานฉบับนี้รู้ข้อนั้น และตั้งชื่อสิ่งที่ทำได้ไว้สามอย่าง ทั้งสามอย่างเป็นธรรมชาติ

สิ่งที่กรณีนี้สอน: นับชั้นป้องกันแล้ว ให้เขียนข้างแต่ละชั้นว่าอะไรทำให้มันล้มพร้อมชั้นอื่น — สองชั้นที่ล้มด้วยเหตุเดียวกัน นับเป็นชั้นเดียว

ชีสสวิส · 1990

แผ่นชีสเรียงซ้อนกันโดยแต่ละแผ่นมีรู คือสิ่งที่คนจำได้จากแบบจำลองนี้ และเป็นสิ่งที่มาถึงหลังสุด ข้อเสนอที่อยู่ใต้ภาพนั้นไม่ใช่เรื่องของแผ่นกั้น มันเป็นเรื่องของเวลา

วันที่ 12 เมษายน 1990 Philosophical Transactions of the Royal Society B เล่ม 327 พิมพ์บทความของ James Reason เรื่อง "The contribution of latent human failures to the breakdown of complex systems" หน้า 475–484 ชื่อบทความคือข้ออ้างทั้งหมด เส้นแบ่งที่เขาเสนอคือระหว่าง active failures ที่ปลายแหลม เห็นทันทีกับ latent conditions ที่ถูกวางไว้นานก่อนหน้าโดยคำตัดสินใจ แบบ และสภาพองค์กร แล้วนอนนิ่งรอ ในปีเดียวกัน Cambridge University Press พิมพ์ Human Error ของเขา ฉบับที่ติดป้ายแบบจำลองนี้ด้วยคำว่า defence in depth อย่างชัดแจ้ง

แบบจำลองไม่ได้เกิดในปีเดียว Justin Larouzée และ Jean-Christophe Le Coze สืบว่ามันถูกพัฒนาต่อเนื่องราวสิบปีเป็นหลายขั้น และได้รับอิทธิพลโดยตรงจาก John Wreathall ภาพที่ทุกคนจำได้จึงเป็นผลปลายทางของกระบวนการนั้น ไม่ใช่จุดตั้งต้น

ข้ออ้างของแบบจำลองมีเท่านี้ แต่ละแผ่นคือแนวป้องกันหนึ่งชั้นที่หยุดเหตุไม่พึงประสงค์ได้ รูคือจุดอ่อนของแต่ละชั้น และรูเหล่านั้นเปลี่ยนขนาดกับตำแหน่งอยู่ตลอด ระบบล้มเมื่อรูในหลายแผ่น "momentarily align" เปิดเส้นทางที่ Reason เรียกว่า trajectory of accident opportunity ความบกพร่องในชั้นเดียวไม่ทำให้อันตรายเกิดจริง เพราะยังมีชั้นอื่น แบบจำลองนี้จึงเป็นข้อโต้แย้งต่อจุดล้มเดี่ยว

คำที่แบกน้ำหนักทั้งภาพคือ momentarily รูเรียงตรงกันเพราะบังเอิญ และมันขยับเอง

คำวิจารณ์ถูกรวบรวมไว้ใน Safety Science เล่ม 126 ปี 2020 โดย Larouzée และ Le Coze มีสี่ข้อหลัก มันถูกจัดอยู่ในตระกูลแบบจำลองเชิงเส้น และถูกมองว่าไม่เป็นระบบ ลดทอนสาเหตุของอุบัติเหตุจนง่ายเกิน มันบอกเป็นนัยว่าอันตรายเดินเป็นเส้นตรงผ่านแผ่นกั้นที่เรียงเป็นระเบียบ มันไม่ให้กลไก — รูไม่มีคำอธิบายว่าเกิดจากอะไร เชื่อมกับปัจจัยองค์กรหรือปัจเจกอย่างไร และข้อที่หนักที่สุด "the weakness of the SCM is that it is irrefutable by standard scientific methods"

Reason เองก็กังวลว่ามันถูกใช้กว้างและตายตัวเกินไป จนคนไล่หาสาเหตุที่ห่างจากเหตุการณ์ทั้งเวลาและสถานที่มากเกินควร ผู้วิจารณ์ทั้งสองไม่ได้ทิ้งมัน พวกเขาสรุปว่ามันยังใช้ได้เพราะฐานคิดเชิงระบบและการใช้งานต่อเนื่องในอุตสาหกรรมความเสี่ยงสูง แล้วเรียกร้องทางเลือกที่รวมข้อมูลเชิงประจักษ์ งานภาคปฏิบัติ และภาพเข้าด้วยกัน

รูที่ขยับเองแล้วมาเรียงตรงกันในวันหนึ่ง คือสิ่งที่แบบจำลองนี้อธิบายได้ ส่วนรูที่ถูกขยับ มันไม่มีคำเรียก

สิ่งที่กรณีนี้สอน: วาดแผ่นชีสเมื่อไร ให้เขียนข้างแต่ละรูว่าอะไรทำให้มันขยับ — รูที่ไม่มีสาเหตุเขียนกำกับคือภาพวาด ไม่ใช่แบบจำลอง

อุบัติเหตุปกติ · 1984

อุบัติเหตุมักจบลงด้วยการหาคนที่ทำผิดในนาทีนั้น ข้อเสนอของ Charles Perrow คืออุบัติเหตุบางประเภทไม่มีนาทีนั้นให้หา เพราะรูปร่างของระบบเป็นตัวผลิตมันขึ้นมาเอง

ในปี 1984 Basic Books พิมพ์ Normal Accidents: Living with High-Risk Technologies ของเขา นักสังคมวิทยาแห่ง Yale ห้าปีหลังอุบัติเหตุ Three Mile Island ปี 1979 ที่เขาบรรยายว่า "unexpected, incomprehensible, uncontrollable and unavoidable"

ข้อเสนอมีสองแกน ประโยคของเขาเอง: "If interactive complexity and tight coupling — system characteristics — inevitably will produce an accident, I believe we are justified in calling it a normal accident, or a system accident"

แกนแรก interactive complexity ระบบที่ความขัดข้องสองจุดขึ้นไปทำปฏิกิริยากันในแบบที่ผู้ออกแบบไม่ได้ตั้งใจและไม่ได้คาด ผู้ควบคุมมองไม่ออกในทันที แกนที่สอง tight coupling ชิ้นส่วนพึ่งพากันแน่นและผูกกันด้วยเวลา จุดหนึ่งเปลี่ยน จุดอื่นเปลี่ยนตามเร็ว จนระบบไม่มีโอกาสฟื้น ถ้อยคำนิยามสองข้อนี้เรียบเรียงจากแหล่งรอง ไม่ใช่ประโยคจากหน้าหนังสือ เงื่อนไขที่สามคือศักยภาพหายนะ ระบบที่ "take the lives of hundreds of people in one blow" ได้

จุดที่สำคัญกับเรื่องส่วนเผื่อคือข้อนี้ ความซับซ้อนอย่างเดียวรอดได้ ปฏิกิริยาที่ไม่คาดคิดระหว่างความขัดข้องไม่กลายเป็น normal accident เมื่อมีเวลาพอจะแก้ก่อนถึงจุดวิกฤต ส่วนเผื่อของ Perrow คือเวลา และ tight coupling คือสิ่งที่เอาเวลาไป

เขาไปไกลกว่านั้นในปี 2004 ตามที่ Scott Sagan สรุปใน Organization & Environment เดือนมีนาคมปีเดียวกัน ว่าความซ้ำซ้อนลดความน่าเชื่อถือได้สามทาง: อุปกรณ์นิรภัยสำรองทำให้ระบบซับซ้อนขึ้นจึงพลาดง่ายขึ้น ความซ้ำซ้อนทำให้คนโยนความรับผิดชอบให้กัน และมันเปิดทางให้แรงกดดันด้านผลผลิตดันระบบให้เดินเร็วขึ้นแต่ปลอดภัยน้อยลง ข้อสรุปเชิงนโยบายของเขาคือระบบที่ซับซ้อน ผูกแน่น และมีศักยภาพหายนะ ควรถูกออกแบบใหม่ทั้งหมด หรือเลิกใช้

ข้อสรุปแบบนั้นมีคนค้าน และข้อถกเถียงยาวที่ตามมาตั้งอยู่บนสมมติฐานที่ไม่มีฝ่ายไหนหยิบขึ้นมาตรวจ คือความขัดข้องมาถึงโดยไม่มีใครเลือกให้มันมา คำถามที่เหลือจึงมีแค่ว่าองค์กรจับมันทันหรือไม่

ฝ่ายค้านมาจาก Berkeley Todd LaPorte, Gene Rochlin และ Karlene Roberts ศึกษาองค์กรที่เดินเทคโนโลยีอันตรายโดยมีอุบัติเหตุน้อยกว่าที่ทฤษฎีทำนาย เรือบรรทุกเครื่องบิน หอควบคุมจราจรทางอากาศ โรงไฟฟ้านิวเคลียร์ บทความของ LaPorte กับ Consolini ปี 1991 ชื่อ "Working in Practice But Not in Theory" และชื่อนั้นคือข้อเสนอทั้งหมด

Scott Sagan ทดสอบสองทฤษฎีกับระบบบัญชาการอาวุธนิวเคลียร์สหรัฐใน The Limits of Safety ปี 1993 LaPorte ตอบในปี 1994 ด้วยบทความชื่อ "A Strawman Speaks Up" และสองฝ่ายยังเถียงกันด้วยว่าระบบนั้นซับซ้อนหรือเป็นเส้นตรง

ทางตันเป็นเรื่องวิธี Perrow ชี้ว่าฝ่าย HRO เลือกตัวอย่างจากผลลัพธ์ คือศึกษาแต่องค์กรที่ยังไม่ล้ม ฝ่ายตรงข้ามชี้กลับว่าทฤษฎีของ Perrow อธิบายความล้มเหลวได้ทุกครั้งเมื่อมองย้อนหลัง กรณีศึกษาของแต่ละฝ่ายมีค่าผลลัพธ์เดียวกันหมด จึงพิสูจน์ผิดไม่ได้ทั้งคู่

Shrivastava, Sonpar และ Pazzaglia เสนอในปี 2009 ว่าสองทฤษฎีมองปรากฏการณ์เดียวกันคนละช่วงเวลา หนังสือเล่มนี้ถูกอ้างอิงใน SSCI/SCI กว่า 1,000 ครั้งภายในปี 2003 และ Nick Pidgeon ประเมินมันใน Nature ปี 2011 สถานะปัจจุบันคือยังไม่จบ

สองฝ่ายเถียงกันเรื่องเดียว คือองค์กรจับความขัดข้องทันหรือไม่ ไม่มีฝ่ายไหนถามว่าถ้าความขัดข้องเลือกเวลามาเอง เวลาที่เป็นส่วนเผื่อจะเหลือเท่าไร

สิ่งที่กรณีนี้สอน: ก่อนเพิ่มอุปกรณ์สำรองชิ้นใด ให้เขียนว่ามันเพิ่มปฏิกิริยาระหว่างชิ้นส่วนกี่คู่ และเอาเวลาไปจากขั้นไหน — ถ้ามันผูกระบบแน่นขึ้น มันไม่ใช่ส่วนเผื่อ

ท่าที่เอาไปใช้ได้: ทุกตัวเลขส่วนเผื่อที่เจอ ให้ตามไปหาเอกสารที่ตั้งมัน ไม่ใช่ตำราที่อ้างมัน แล้วอ่านหาสองอย่าง: มันเผื่อรายการไหนของความไม่รู้ และมันถูกแบ่งมาจากเลขเดิมหรือถูก derive ใหม่

จะพลิกคำตัดสินของภาคนี้ได้เมื่อ: มีใครแสดงบันทึกการปรับเทียบของโค้ด โครงสร้างระดับชาติฉบับใดฉบับหนึ่ง ที่ตัวคูณย่อยถูก derive จากสถิติการวิบัติจริง และค่าที่ได้ไม่ทวนซ้ำตัวคูณรวมของโค้ดรุ่นก่อน

แหล่งของภาคนี้:

- Wikipedia, Factor of safety — https://en.wikipedia.org/wiki/Factor_of_safety
- Hansson, Safety Factors, Encyclopedia of Science, Technology, and Ethics — https://www.encyclopedia.com/science/encyclopedias-almanacs-transcripts-and-maps/safety-factors
- Beal, A history of the safety factors, The Structural Engineer 89(20), 2011 — https://anbeal.co.uk/safetyfactorhistory.html
- Beal, Factors of Ignorance?, The Structural Engineer 79(20), 2001 — https://www.istructe.org/journal/volumes/volume-79-(published-in-2001)/issue-20/factors-of-ignorance/
- Pugsley, Concepts of Safety in Structural Engineering, 1951 — https://www.icevirtuallibrary.com/doi/abs/10.1680/ijoti.1951.12755
- Rankine, A Manual of Applied Mechanics, 1858 — https://archive.org/details/manualappmecha00rankrich
- 14 CFR 25.303, Factor of safety — https://www.ecfr.gov/current/title-14/chapter-I/subchapter-C/part-25/subpart-C/subject-group-ECFRea44cbb3048b817/section-25.303
- Zipay, Modlin & Larsen, The Ultimate Factor of Safety for Aircraft and Spacecraft, NASA — https://ntrs.nasa.gov/api/citations/20150003482/downloads/20150003482.pdf
- Gray, Elisha Otis' "Improved Elevator", Elevator World — https://elevatorworld.com/article/elisha-otis-improved-elevator/
- Gray, Elisha Otis at Crystal Palace, part 2, Elevator World — https://elevatorworld.com/article/elisha-otis-at-crystal-palace-part-2/
- EDN, 1st commercial elevator starts running, March 23, 1857 — https://www.edn.com/1st-commercial-elevator-starts-running-march-23-1857/
- Pierce, The Evolution of Structural Design Specifications in the United States, STRUCTURE, 2015 — https://www.structuremag.org/article/the-evolution-of-structural-design-specifications-in-the-united-states/
- Greene, Decoded: Fail Safe vs. Fail Secure – When and Where? — https://idighardware.com/2023/10/decoded-fail-safe-vs-fail-secure-when-and-where/
- Greene, Decoded: Quick Tips – Delayed Egress — https://idighardware.com/2014/10/decoded-quick-tips-delayed-egress/

- Henderson Engineers, Door Locking Requirements in NFPA 101 — https://www.hendersonengineers.com/insight_article/door-locking-requirements-in-nfpa-101/
- IAEA INSAG-10, Defence in Depth in Nuclear Safety, 1996 — https://www-pub.iaea.org/MTCD/Publications/PDF/Pub1013e_web.pdf
- Wellock, Defense in Depth – A War for Safety, NRC — https://www.nrc.gov/reading-rm/basic-ref/students/history-101/defense-in-depth
- Wikipedia, Defence in depth (non-military) — https://en.wikipedia.org/wiki/Defence_in_depth_(non-military)
- Reason, The contribution of latent human failures to the breakdown of complex systems, 1990 — https://royalsocietypublishing.org/doi/10.1098/rstb.1990.0090
- Larouzée & Le Coze, Good and bad reasons: The Swiss cheese model and its critics, Safety Science 126, 2020 — https://www.sciencedirect.com/science/article/pii/S0925753520300576
- Larouzée & Le Coze, open-access deposit — https://ineris.hal.science/ineris-03318305v1/document
- Wikipedia, Normal Accidents — https://en.wikipedia.org/wiki/Normal_Accidents
- Perrow, Normal Accidents, Princeton University Press — https://www.jstor.org/stable/j.ctt7srgf
- LaPorte & Consolini, Working in Practice But Not in Theory, JPART 1(1), 1991 — https://academic.oup.com/jpart/article-abstract/1/1/19/978459
- Sagan, The Limits of Safety, 1993 — https://press.princeton.edu/books/paperback/9780691021010/the-limits-of-safety
- Sagan, Learning from Normal Accidents, Organization & Environment, 2004 — https://web.archive.org/web/20040714202943/http://iis-db.stanford.edu/pubs/20276/sagan_oe_dec03.pdf
- Shrivastava, Sonpar & Pazzaglia, Normal Accident Theory versus High Reliability Theory, Human Relations, 2009 — https://journals.sagepub.com/doi/10.1177/0018726709339117
- Pidgeon, In retrospect: Normal accidents, Nature 477, 2011 — https://doi.org/10.1038/477404a

2 · เผื่อแล้วยังพัง

เก้ากรณีในภาคนี้เผื่อไว้ทุกกรณี ค่าความปลอดภัยถูกคำนวณจริง บางกรณีเกินที่กฎหมายบังคับ และทุกกรณีพัง

คำอธิบายอยู่ในนิยามของตัวเลขนั้นเอง ค่าความปลอดภัยคืออัตราส่วนระหว่างกำลังที่มีกับแรงที่ถูกตั้งชื่อไว้ในแบบ ตั้งชื่อแรงผิด อัตราส่วนนั้นก็เป็นแค่ตัวเลขประดับ มันไม่ได้ผิดเพราะใครคำนวณผิด มันผิดเพราะมันตอบคำถามอื่น

ภาคนี้จึงถามคำถามเดียวซ้ำเก้าครั้ง ส่วนเผื่อถูกคิดเทียบกับอะไร และอะไรคือสิ่งที่เข้าโจมตีโครงสร้างจริง

ทาโคมาแนโรวส์ · 7 พฤศจิกายน 1940

ในปี 1940 วิชาสะพานแขวนเชื่อว่าตัวเองแก้ปัญหาลมได้แล้ว ไม่ใช่ด้วยการสร้างให้หนาขึ้น แต่ด้วยการค้นพบว่าไม่ต้องหนาขนาดนั้น deflection theory ของ Leon Moisseiff ถือว่าน้ำหนักตัวสะพานกับรูปทรงเคเบิลรับแรงด้านข้างได้เอง โครงเสริมความแข็งจึงลดลงได้ สะพานที่บางกว่าเดิมไม่ใช่ความประมาท มันคือสิ่งที่ทฤษฎีอนุญาต

วันที่ 7 พฤศจิกายน 1940 สี่เดือนหลังเปิดใช้ สะพานแขวนที่พื้นลึกเพียง 1 ใน 350 ของช่วงกลางบิดตัวจนพื้นช่วงกลางขาดลงช่องแคบ ไม่มีคนตาย มีสุนัขชื่อ Tubby ตายในรถบนสะพานหนึ่งตัว โครงสร้างเหนือน้ำทั้งหมดถูกตีราคาเป็นเศษเหล็ก เหลือแต่ตอม่อสองต้นกับฐานยึดเคเบิลที่ไม่เสียหาย

ค่าความปลอดภัยของสะพานนี้ไม่ได้ต่ำ มันตั้งฉากกับสิ่งที่ฆ่าสะพาน ส่วนเผื่อถูกคิดกับลมในฐานะแรงดันสถิต คือลมกดด้านข้างแรงเท่าไร โครงรับได้เท่าไร คูณตัวเลขนั้นอีกสามเท่าก็ไม่ช่วย เพราะแรงที่ทำให้พื้นขาดไม่ได้มาจากลม

Moisseiff แทนที่ stiffening truss สูง 25 ฟุตในแบบเดิมของ Clark Eldridge ด้วยคานแผ่นเหล็กตันสูง 8 ฟุต ความกว้างต่อช่วงอยู่ที่ 1 ต่อ 72 คานแผ่นตันรับแรงกดด้านข้างได้ดี แต่ต้านการบิดตัวได้แทบไม่มี ไม่มีใครต้องคิดถึงข้อนี้ ตราบใดที่แรงที่คิด

อยู่คือแรงกด ความเร็วลมออกแบบที่อ้างกันในเอกสารรอง เล่มนี้ยังไม่ได้ตรวจกับเอกสารต้นฉบับ จึงไม่พิมพ์

ลมวันนั้นอยู่ในกรอบที่แบบรับได้ ตัวเลขความเร็วที่รายงานกันทั่วไปยังไม่ได้ตรวจกับบันทึกเครื่องวัดลมต้นทาง หลังสิบโมงเช้าไม่นาน แถบรัดเคเบิลกลางช่วงบนเคเบิลด้านเหนือลื่นหลุด เคเบิลแยกเป็นสองท่อนยาวไม่เท่ากัน ความไม่สมดุลถ่ายเข้าคานแผ่นที่แทบไม่มีความแข็งต่อการบิด และการแกว่งเปลี่ยนจากขึ้นลงเป็นบิดตัว

จากจุดนั้นเรื่องเปลี่ยนชนิด พื้นสะพานเองเป็นผู้สร้างแรงที่ขยายการเคลื่อนไหวของมัน ยิ่งบิดมาก แรงที่มันสร้างขึ้นก็ยิ่งมาก ลมไม่ใช่ตัวขับจากภายนอกอีกต่อไป มันเป็นแค่แหล่งพลังงานที่โครงสร้างดึงเข้ามาใช้ Billah และ Scanlan เรียกสภาพนี้ว่า negative damping ในทิศทางการบิด ระบบที่ปกติกินพลังงานของตัวเองจนสงบ กลับเติมพลังงานให้ตัวเองแทน

ฉบับที่ตำราฟิสิกส์เล่ากันว่าสะพานพังเพราะ resonance — ลมพัดตรงกับความถี่ธรรมชาติ — ไม่ถูก และบทความของ Billah กับ Scanlan ใน American Journal of Physics ปี 1991 เขียนขึ้นเพื่อแก้เรื่องนี้โดยเฉพาะ resonance ต้องมีตัวขับภายนอกที่ความถี่ตรงกัน ถ้าไม่ตรงก็ไม่เกิด ส่วน torsional flutter ไม่ต้องมีความถี่ให้ตรง โครงสร้างขับตัวเอง เรียกชื่อผิด ก็จะป้องกันผิดทาง คณะกรรมการ Carmody รายงานเดือนมีนาคม 1941 ว่าสาเหตุหลักคือความยืดหยุ่นเกิน และ WSDOT เองยังเขียนว่าสาเหตุแน่ชัดไม่เป็นที่ยุติ

กรณีนี้ไม่ใช่เรื่องใหม่แม้แต่ตอนนั้น J. K. Finch เขียนใน Engineering News-Record สี่เดือนหลังสะพานพังว่าสะพานแขวนพังเพราะลมมาแล้วทั้งศตวรรษ ครั้งล่าสุดคือ Niagara-Clifton ปี 1889 สิ่งที่หายไปคือความจำ ห้าสิบปีที่ไม่มีสะพานพังถูกอ่านว่าเป็นหลักฐานว่าปลอดภัย

สิ่งที่กรณีนี้สอน: ถามว่าค่าความปลอดภัยของคุณคิดกับแรงหรือกับกลไก — ถ้าโครงสร้างสร้างแรงขึ้นเองได้ ตัวคูณของแรงภายนอกไม่มีความหมาย

เดอ ฮาวิลแลนด์ คอเม็ต · 1954

เครื่องบินโดยสารที่บินสูงต้องอัดความดันในลำตัวไว้ให้คนข้างในหายใจได้ และลำตัวที่อัดความดันคือภาชนะที่พองขึ้นตอนไต่แล้วยุบลงตอนร่อน หนึ่งเที่ยวบินหนึ่งรอบ คำถามที่วิชานี้ตอบได้ไม่ดีในตอนนั้นคือภาชนะแบบนั้นทนได้กี่รอบก่อนจะร้าว

วันที่ 10 มกราคม 1954 เครื่อง Comet 1 ทะเบียน G-ALYP ของ BOAC แตกกลางอากาศขณะไต่ผ่าน 27,000 ฟุตนอกชายฝั่งเกาะ Elba คนบนเครื่อง 35 คนตายทั้งหมด วิทยุขาดกลางประโยค สิบหกวันหลังฝูงบินกลับมาบินอีกครั้ง วันที่ 8 เมษายน 1954 G-ALYY ของ South African Airways แตกขณะไต่ผ่าน 35,000 ฟุตใกล้ Naples ตาย 21 คน

ส่วนเผื่อของลำตัวเครื่องนี้เกินที่กฎหมายบังคับ และมันถูกวัดบนโลหะผิดชิ้น

ข้อกำหนด BCAR ของอังกฤษให้ออกแบบลำตัวรับความดัน 2P และทดสอบ proof ที่ 1.33P เมื่อ P คือความดันใช้งานราว 8.25 psi de Havilland ไม่ได้ทำแค่ตามนั้น บริษัทออกแบบที่ 2.5P และทดสอบ proof ที่ 2P

เมื่อ ARB เสนอให้ทดสอบความล้าของลำตัวรับความดันในเดือนมิถุนายน 1953 de Havilland อัดความดันลำตัวต้นแบบซ้ำจาก 0 ถึง 1P อีก 16,000 รอบ มันแตกที่รอบที่ 16,000 ตรงมุมหน้าต่างทรงเหลี่ยม เทียบกับอายุปลอดภัยเป้าหมาย 10,000 รอบ ดูเหมือนเผื่อไว้ 60%

ตัวเลขนั้นไม่ได้โกหก มันตอบคำถามเกี่ยวกับลำตัวที่ไม่มีอยู่ในฝูงบิน ลำตัวที่ทนได้ 16,000 รอบคือลำตัวที่เคยผ่าน proof test ที่ 2P มาก่อน ความดันขนาดนั้น cold-work โลหะตรงจุดรวมความเค้นพอดี มุมหน้าต่างของชิ้นทดสอบจึงทนความล้าได้ดีกว่าเครื่องบินทุกลำที่ส่งมอบ การทดสอบพิสูจน์กำลัง ทำให้ชิ้นที่ใช้พิสูจน์อายุแข็งกว่าของจริง

ข้อที่สองอยู่ในปรัชญาการออกแบบ safe-life ให้ชิ้นส่วนเป็นจำนวนรอบที่รับรอง ไม่มีเส้นทางรับแรงสำรอง ไม่มีตัวหยุดรอยร้าว รอยร้าวที่เกิดเร็วกว่ากำหนดจึงไม่ใช่สภาพเสื่อมที่ช่างจะเจอตอนตรวจ มันคือเครื่องบินทั้งลำ

RAE Farnborough แก้ปัญหาการมองไม่เห็นนั้นด้วยวิธีที่ตรงไปตรงมาจนน่าจำ คือจมเครื่องบินทั้งลำ ถังน้ำ 200,000 แกลลอนถูกสร้างครอบลำตัว G-ALYU ทั้งลำ แล้วอัดความดันเป็นรอบด้วยน้ำแทนอากาศ เพื่อให้รอยแตกหยุดอยู่ในถัง วันที่ 24 มิถุนายน 1954 ลำตัวนั้นแตกหลังบินจริง 1,230 เที่ยวบวกรอบในถัง 1,830 รอบ รวมราว 3,060 รอบ จุดเริ่มรอยร้าวคือมุมช่องประตูหนีภัยด้านหน้า

ฉบับที่เล่ากันว่าหน้าต่างผู้โดยสารทรงสี่เหลี่ยมทำให้เครื่องตก ไม่ตรงกับหลักฐาน รอยร้าวของ G-ALYP เริ่มที่ช่องเสาอากาศ ADF บนหลังคาลำตัว รอยร้าวของ G-ALYU เริ่มที่ประตูหนีภัย และจุดเริ่มของ G-ALYY ไม่เคยพบ เครื่องที่ตกที่ Elba บินมาราว 1,290 เที่ยวตามตัวเลขของ FAA เครื่องที่ Naples 900 เที่ยว

ศาลสอบสวนของ Lord Cohen พิมพ์รายงาน CAP 127 วันที่ 1 กุมภาพันธ์ 1955 และ BCAR D3-7 ข้อ 4.3.2 ถูกแก้ให้ทดสอบความล้าบนลำตัวที่ไม่เคยใช้ทดสอบกำลังมาก่อน ปิดช่องโหว่ cold-working ด้วยชื่อของมันเอง อุตสาหกรรมย้ายไป fail-safe และต่อมาเป็น damage tolerance

สิ่งที่กรณีนี้สอน: ตรวจว่าชิ้นที่ใช้วัดส่วนเผื่อผ่านอะไรมาก่อนถูกวัด — ชิ้นทดสอบที่แข็งกว่าของจริงให้ตัวเลขที่ของจริงชิ้นไหนก็ใช้ไม่ได้

ไททานิก · 15 เมษายน 1912

คำถามที่ Board of Trade ตอบในปี 1894 ไม่ใช่คำถามที่คนสมัยนี้คิด เขาไม่ได้ถามว่าจะเอาคนทั้งลำลงน้ำอย่างไร แต่ถามว่าจะพาคนจากเรือที่กำลังจมไปถึงเรือที่แล่นมาช่วยอย่างไร คณะกรรมการปี 1886 ให้เหตุผลว่าเรือชูชีพกลางมหาสมุทรช่วยชีวิตได้น้อย เว้นแต่มีเรือลำอื่นอยู่ใกล้ และคณะกรรมการปี 1887 ตั้งเป้าที่ "อากาศปานกลาง" เรือชูชีพในกฎนี้จึงเป็นรถรับส่งไปเรือกู้ภัย ไม่ใช่ที่นั่งสำรองสำหรับทุกคน

คืนวันที่ 14 เมษายน 1912 เรือที่มีคน 2,201 คนบนเรือตามรายงานของ Lord Mersey ชนภูเขาน้ำแข็ง และจมลงในเช้ามืดวันที่ 15 เรือลำนี้มีเรือชูชีพรองรับได้ 1,178 คน ตัวเลขนั้นเกินที่กฎหมายบังคับ

ประโยคสุดท้ายคือหัวใจของกรณีนี้ ฉบับที่ว่าเจ้าของเรือตัดงบเรือชูชีพหรือแหกกฎผิด เรือลำนี้ทำเกินกฎ 22% แล้วยังไม่มีที่นั่งให้คนเกือบครึ่งลำ

กฎที่ใช้คือตารางของ Board of Trade ออกตาม Merchant Shipping Act 1894 ลงวันที่ 9 มีนาคม 1894 ตารางนี้ผูกจำนวนเรือชูชีพกับระวางตันรวม ไม่ใช่กับจำนวนคนบนเรือ เรือที่หนักขึ้นจึงต้องมีเรือชูชีพมากขึ้น ส่วนเรือที่บรรทุกคนมากขึ้นโดยระวางตันเท่าเดิม ไม่ต้องเพิ่ม

ตารางนี้หยุดอยู่ที่ขั้นบนสุด "10,000 ตันขึ้นไป" เรือทุกลำในขั้นนั้นต้องมีเรือชูชีพใต้ davit 16 ลำ รวม 5,500 ลูกบาศก์ฟุต เรือขนผู้อพยพต้องเพิ่มอีกสามในสี่ เป็น 9,625 ลูกบาศก์ฟุต ที่อัตรา 10 ลูกบาศก์ฟุตต่อคนตามที่คณะสอบสวนใช้ นั่นคือ 962 คน

ไททานิกมีเรือชูชีพ 20 ลำเทียบกับที่บังคับ 16 รายงานของ Wreck Commissioner เขียนว่าความจุ 1,178 คนนั้นเกินข้อกำหนดของตารางและกฎไปไกล เกินจนไม่เคยต้องใช้ Rule 12 ข้อลดหย่อนสำหรับเรือที่แบ่งห้องกันน้ำได้ตามมาตรฐาน ที่จะลดข้อกำหนดลงเหลือราว 756 คน

ตัวเลขผู้เสียชีวิตที่ยกกันทั่วไปคือราว 1,500 เล่มนี้ยืนยันตัวเลขนั้นกับต้นทางไม่ได้

ส่วนเผื่อจึงถูกคิดเทียบกับระวางตัน ไม่ใช่กับจำนวนชีวิตบนเรือ สิ่งที่โจมตีกฎคือขนาดที่เลื่อนไป ตอนเขียนตาราง เรือขนผู้อพยพลำใหญ่ที่สุดคือ Lucania 12,952 ตัน ขั้นบนสุดจึงพอดีกับโลกในปี 1894 ส่วนไททานิก 46,328 ตัน คือ 3.6 เท่าของขั้นบันไดสุดท้ายที่ไม่มีขั้นต่อ บันไดไม่ได้พัง มันแค่สั้นกว่าตึกที่คนสร้างขึ้นมาทีหลัง

Chalmers ที่ปรึกษาด้านการเดินเรือของ Board of Trade ปี 1896–1911 ให้การว่าทำไมเขาไม่แก้ เหตุผลของเขาไม่ได้อ่อน สถิติอุบัติเหตุสะอาด เรือแข็งแรงและแบ่งห้องดีขึ้น มีโทรเลขไร้สาย เรือชูชีพเกิน 16 ลำจัดการกลางทะเลไม่ทัน และเจ้าของเรือก็เผื่อเกินกฎกันเองอยู่แล้ว

ทุกข้อในรายการนั้นมีสิ่งหนึ่งเหมือนกัน มันเป็นเหตุผลว่าฉากเหตุการณ์จะไม่เกิด ไม่มีข้อไหนเป็นเหตุผลว่าถ้าเกิดแล้วส่วนเผื่อจะพอ Mersey เขียนว่าเห็นน้ำหนักของคำอธิบายนี้ แต่มันไม่ทำให้ความล่าช้าชอบธรรม

การแก้กฎเริ่มแล้วและช้าเกินไป วันที่ 28 กุมภาพันธ์ 1911 Archer หัวหน้าผู้ตรวจเรือของ Board of Trade ร่างตารางใหม่ที่ให้เรือขนาดไททานิกมีความจุ 24,937 ลูกบาศก์ฟุต พอสำหรับ 2,201 คนในคืนนั้น กฎยังไม่เปลี่ยนตอนเรือออกจากท่า

สิ่งที่กรณีนี้สอน: หาเพดานของตารางที่กฎของคุณอ้าง แล้วเทียบกับขนาดของสิ่งที่คุณสร้างวันนี้ ไม่ใช่ขนาดของวันที่ตารางถูกเขียน

ทางเดินลอยของ Hyatt Regency · 17 กรกฎาคม 1981

ทางเดินลอยมีปัญหาที่พื้นธรรมดาไม่มี คือไม่มีอะไรอยู่ข้างล่างมันเลย น้ำหนักทุกกรัมต้องเดินขึ้นไปหาหลังคา และเส้นทางเดียวที่พาน้ำหนักขึ้นไปได้คือแท่งเหล็กที่ห้อยลงมา วันที่ 17 กรกฎาคม 1981 ระหว่างงานเต้นรำในห้องโถงโรงแรม Hyatt Regency เมือง Kansas City เส้นทางนั้นขาด ทางเดินลอยสองชั้นถล่มลงมาทับคนข้างล่าง ตาย 114 คน บาดเจ็บกว่า 200 คน เป็นโครงสร้างถล่มที่ตายมากที่สุดในอเมริกาจนถึงปี 2001

ส่วนเผื่อของกรณีนี้ไม่ได้คิดผิดแกน มันไม่เคยถูกคิดเลยสำหรับจุดต่อแบบที่สร้างจริง ไม่มีลมพิเศษ ไม่มีความล้า ไม่มีพลศาสตร์ ฝูงชนบนทางเดินคือน้ำหนักจรธรรมดาที่ code คาดไว้แล้ว สิ่งเดียวที่ไม่ธรรมดาคือจุดต่อที่รับน้ำหนักนั้น ไม่ใช่จุดต่อที่ใครเคยคำนวณ

แบบเดิมแขวนทางเดินชั้นสี่และชั้นสองด้วยแท่งเหล็กเส้นเดียวต่อเนื่อง แท่งทะลุคานกล่องของทางเดินชั้นสี่ มีนอตกับแหวนรองรับตรงนั้น แล้วยาวต่อลงไปสิ้นสุดใต้คานกล่องชั้นสอง ลองไล่ตามน้ำหนักดูทีละช่วง คนที่ยืนบนชั้นสองกดคานกล่องชั้นสอง คานกล่องกดนอตตัวล่าง นอตตัวล่างดึงแท่ง แล้วแท่งพาน้ำหนักก้อนนั้นวิ่งขึ้นไปหาหลังคาโดยตรง ผ่านชั้นสี่ไปเฉยๆ จุดต่อที่ชั้นสี่จึงรับน้ำหนักเฉพาะทางเดินชั้นสี่

แบบที่สร้างจริงใช้แท่งสั้นสองแท่ง แท่งหนึ่งจากเพดานถึงชั้นสี่ อีกแท่งจากชั้นสี่ลงชั้นสอง เหตุผลที่บันทึกไว้ธรรมดาจนน่าเสียดาย ผู้ผลิต Havens ไม่อยากทำเกลียวตลอดความยาวแท่งยาวเพื่อขันนอตเข้าไปถึงกลางแท่ง

การเปลี่ยนนี้ตัดเส้นทางที่น้ำหนักชั้นสองเคยใช้ทิ้ง น้ำหนักชั้นสองไม่วิ่งขึ้นหลังคาเองอีกแล้ว มันถูกส่งเข้าคานกล่องชั้นสี่ก่อน แล้วให้คานกล่องชั้นสี่แบกทั้งสองชั้นขึ้นไป น้ำหนักที่จุดนั้นเพิ่มเป็นสองเท่า จากการแก้แบบที่บนกระดาษดูเหมือนเรื่องวิธีผลิตล้วนๆ

National Bureau of Standards วัดได้ว่าน้ำหนักสูงสุดที่จุดต่อคานกล่องกับแท่งแขวนชั้นสี่ตอนถล่มอยู่ที่ 53% ของน้ำหนักออกแบบที่ Kansas City Building Code กำหนด จุดต่อพังที่ครึ่งหนึ่งของสิ่งที่ code บอกว่ามันต้องรับได้

ฉบับที่เล่ากันว่าแบบเดิมปลอดภัยแล้วผู้รับเหมาทำพัง ไม่ตรงกับหลักฐานทั้งสองข้าง วรรณกรรมวิศวกรรมระบุตรงกันว่าจุดต่อแบบเดิมก็ต่ำกว่า code อยู่แล้ว ส่วนสัดส่วนที่แน่นอนเล่มนี้ยังไม่ได้อ่านจากรายงาน NBS โดยตรง จึงไม่พิมพ์ การเปลี่ยนแบบจึงไม่ได้เปลี่ยนของดีให้เป็นของเสีย มันเอาความจุที่ไม่พออยู่แล้วมาหารสอง

และการแก้แบบของผู้ผลิตผ่านการอนุมัติโดยไม่มีใครคำนวณจุดต่อใหม่ ตราวิศวกรผู้รับผิดชอบไปอยู่บนแบบที่ไม่มีใครวิเคราะห์

เดือนกุมภาพันธ์ 1984 อัยการสูงสุดรัฐ Missouri ยื่นคำร้องให้ลงโทษวิศวกรสองคนกับบริษัทของพวกเขา เดือนพฤศจิกายน 1985 ผู้พิพากษาชั้นปกครองตัดสินว่ามีความผิดฐาน gross negligence ปี 1986 คณะกรรมการวิศวกรรมของรัฐเพิกถอนใบอนุญาตของ Daniel M. Duncan และ Jack D. Gillum และเพิกถอนใบรับรองของบริษัท ไม่มีคดีอาญา

สิ่งที่กรณีนี้สอน: ทุกครั้งที่จุดต่อเปลี่ยน ให้ถามว่าใครคำนวณมันใหม่ และขอดูตัวเลข — ตราบนแบบไม่ใช่การคำนวณ

แชลเลนเจอร์ · 28 มกราคม 1986

จรวดเชื้อเพลิงแข็งประกอบขึ้นจากท่อนที่ต่อกัน และรอยต่อทุกรอยคือช่องที่ก๊าซร้อนอยากผ่าน ของที่กันมันไว้คือยางเส้นหนึ่ง งานของยางเส้นนั้นไม่ใช่การทนแรง แต่คือการดีดกลับให้ทัน

เวลา 11:38 น. ตามเวลาฝั่งตะวันออกของวันที่ 28 มกราคม 1986 กระสวยอวกาศ Challenger ขึ้นจากแท่นที่อุณหภูมิ 36°F และแตกออกในวินาทีที่ 73 ลูกเรือเจ็ดคนตายทั้งหมด — Dick Scobee, Michael Smith, Judith Resnik, Ronald McNair, Ellison Onizuka, Gregory Jarvis และ Christa McAuliffe ครูจากโครงการ Teacher in Space

คณะกรรมการ Rogers สรุปว่าสาเหตุคือซีลใน aft field joint ของจรวดเชื้อเพลิงแข็งด้านขวาล้มเหลว ปล่อยให้ก๊าซร้อนรั่วผ่านรอยต่อระหว่างเผาไหม้ และเรียกแบบรอยต่อนั้นว่าออกแบบผิดพลาดและอ่อนไหวต่อปัจจัยหลายอย่างเกินรับได้ อุณหภูมิเป็นหนึ่งในนั้น

ค่าความปลอดภัยทุกตัวของจรวดลำนั้นยังอยู่ครบและไม่ได้ถูกใช้เลย ความดันแตกกำลังปลอกจรวด ค่าความปลอดภัยด้านโครงสร้าง ไม่มีตัวไหนเกี่ยว เพราะส่วนเผื่อของรอยต่อถูกคิดในฐานะซีลความดันสถิต คือถามว่ายางทนความดันได้เท่าไร ทั้งที่คำถามจริงคือยางกลับเข้าที่ได้เร็วแค่ไหน

ตอนจุดจรวด รอยต่อถ่างออก และ O-ring ต้องดีดตามไปปิดช่องว่างนั้นให้ทัน ความสามารถดีดกลับนั้นเรียกว่า resiliency และ resiliency ขึ้นกับอุณหภูมิ ยางที่เย็นลงจะช้าลง ส่วนรอยต่อที่ถ่างออกตอนจุดจรวดไม่รอ ส่วนเผื่อที่มีความหมายในวันนั้นจึงเป็นเวลาตอบสนองของยางในหน่วยมิลลิวินาที ไม่ใช่ความดันที่มันทนได้

36°F อยู่นอกซองที่รับรอง และเย็นกว่าการปล่อยครั้งไหนก่อนหน้าราวสิบห้าองศา ครั้งที่เย็นที่สุดก่อนหน้าคือ STS-51-C เดือนมกราคม 1985 ที่ 53°F ซึ่งพบ O-ring สึกกร่อนเมื่อกู้จรวดขึ้นมา

ทุกเที่ยวบินที่มีการสึกกร่อนถูกอ่านว่ารอยต่อทนการสึกกร่อนได้ ไม่ใช่ว่าส่วนเผื่อลดลง หลักฐานชุดเดียวกันอ่านได้สองแบบ และองค์กรเลือกแบบที่ให้บินต่อ Diane Vaughan ตั้งชื่อรูปแบบนี้ภายหลังว่า normalization of deviance

ตัวเลขทำงานของฝ่ายบริหาร NASA สำหรับโอกาสสูญเสียยานอยู่ที่ราว 1 ใน 100,000 Feynman ในภาคผนวก F ของรายงานประเมินว่าใกล้ 1 ใน 100 และโจมตีวิธีที่ตัวเลขทางการถูกผลิตขึ้น

สิ่งที่เกิดในคืนก่อนปล่อยสรุปได้ในประโยคเดียว ภาระพิสูจน์กลับด้าน วิศวกรถูกขอให้พิสูจน์ว่าบินไม่ปลอดภัย แทนที่จะถูกขอให้พิสูจน์ว่าปลอดภัย

Morton Thiokol เปิดประชุมทางไกลเพื่อยกพยากรณ์อุณหภูมิขึ้นเทียบกับ O-ring วิศวกรของ Thiokol แนะนำไม่ให้ปล่อยต่ำกว่า 53°F ผู้บริหาร NASA ท้าทายคำแนะนำนั้น Thiokol ขอพักหารือภายในราว 30 นาที แล้วผู้บริหารระดับสูงของ Thiokol กลับคำแนะนำของวิศวกรตัวเองและให้ปล่อยได้

ฉบับที่เล่ากันว่า NASA สั่งข้ามหัววิศวกร Thiokol จึงย่อจนผิด NASA ท้าทาย แต่คนที่กลับคำคือ Thiokol เอง คณะกรรมการ Rogers พบว่าการกลับคำไม่ได้มาจากข้อมูลทางวิศวกรรมใหม่ และให้อุบัติเหตุมีสองสาเหตุ ซีลที่ล้มเหลวที่อุณหภูมินอกการรับรอง กับกระบวนการตัดสินใจที่บกพร่องในบทที่ V ของรายงาน

สิ่งที่กรณีนี้สอน: ถ้าเงื่อนไขวันนี้อยู่นอกซองที่เคยทดสอบ ให้เขียนคำถาม "พิสูจน์ว่าปลอดภัย" ลงกระดาษก่อนเข้าประชุม — ไม่ใช่ "พิสูจน์ว่าอันตราย"

เชอร์โนบิล · 26 เมษายน 1986

ที่เชอร์โนบิลมีคนทำในสิ่งที่ควรทำ เขาพบช่องโหว่ในแนวป้องกันของโรงไฟฟ้า ถ้าไฟดับทั้งสถานี จะมีช่วงหนึ่งที่ปั๊มน้ำหล่อเย็นหลักไม่มีไฟเลี้ยง ระหว่างรอเครื่องดีเซลฉุกเฉินขึ้น คำถามคือแรงเฉื่อยของกังหันที่กำลังช้าลงจะปั่นไฟพอกลบช่วงนั้นได้หรือไม่ ขั้นตอนที่ตอบคำถามนี้ชื่อ turbine-generator rundown test และจะรันมันได้ต้องปิดระบบความปลอดภัยที่จะสั่งยกเลิกเสียก่อน

ขั้นตอนที่ไล่หาส่วนเผื่อ คือสิ่งที่ทำลายเตา

เวลา 01:23:40 ของวันที่ 26 เมษายน 1986 ผู้ควบคุมเตาปฏิกรณ์หน่วยที่ 4 กดปุ่ม AZ-5 ปุ่มดับเตาฉุกเฉิน และเตาระเบิดในไม่กี่วินาทีถัดมา

คนงานสองคนตายจากการระเบิดในคืนนั้น acute radiation syndrome ยืนยันใน 134 คนจาก 237 ที่วินิจฉัยเบื้องต้น และ 28 คนในนั้นตายภายในไม่กี่สัปดาห์ รวม 30 ตัวเลข 31 ที่ยกกันทั่วไปนับเพิ่มอีกหนึ่งราย เล่มนี้ยืนยันรายนั้นไม่ได้

ส่วนเผื่อที่ผู้ควบคุมใช้คือ Operating Reactivity Margin หรือ ORM จำนวนแท่งควบคุมเทียบเท่าที่ยังเสียบอยู่ในแกน ขีดล่างคือ 15 แท่ง ผู้ควบคุมดูจะเชื่อว่าถ้าไม่ต่ำกว่า 15 ก็ปลอดภัย แต่ความหมายของ ORM ขึ้นกับตำแหน่งของแท่งและรูปกำลังตามแนวแกน ตัวเลขเดียวกันจึงแปลว่าคนละอย่างได้ ขึ้นกับว่าแกนอยู่ในสภาพไหน มันเป็นตัวแทน ไม่ใช่ตัวเลขเดี่ยว

สิ่งที่โจมตีเตาคือคุณสมบัติสองอย่างของ RBMK ที่ทำงานร่วมกัน อย่างแรก positive void coefficient RBMK ใช้กราไฟต์หน่วงนิวตรอนและใช้น้ำหล่อเย็น เมื่อน้ำเดือดเป็นไอ ตัวดูดกลืนนิวตรอนหายไปแต่กราไฟต์ยังหน่วงอยู่ reactivity จึงขึ้น และกำลังที่ขึ้นทำให้น้ำเดือดมากขึ้นอีก วงจรนี้แรงที่สุดที่กำลังต่ำ และเตาถูกรันที่กำลังต่ำพอดี

อย่างที่สอง ปลายแท่งควบคุมเป็นกราไฟต์ เมื่อแท่งเลื่อนลงในช่องที่เต็มด้วยน้ำ ปลายกราไฟต์ไล่น้ำออกก่อน แทนตัวดูดกลืนด้วยตัวหน่วง การเสียบแท่งจึงเพิ่ม reactivity ที่ก้นแกนก่อนจะลดมัน ด้วยแท่งส่วนใหญ่ถอนออกจนสุด การกด AZ-5 ใส่ reactivity ก้อนใหญ่ลงก้นแกน และผ่านไปกว่าห้าวินาทีก่อน boron carbide จะเบรกอะไรได้

ปุ่มที่เขียนว่าหยุดทำงานถูกต้องตามแบบของมันทุกประการ และสิ่งที่แบบของมันทำในวินาทีแรกคือเร่ง ในสภาพแกนแบบนั้น ปุ่มดับเตาคือตัวจุดชนวน

ฉบับที่ว่าผู้ควบคุมประมาทและแหกกฎคือคำตัดสินทางการ INSAG-1 เดือนสิงหาคม 1986 และ IAEA แก้มันเองใน INSAG-7 ปี 1992 ที่ย้ายน้ำหนักไปที่แบบเตา INSAG-7 ระบุสามสิ่งที่เกิดพร้อมกัน ฟิสิกส์ของเตา แบบของแท่งควบคุม และการที่เตาถูกพาไปยังสภาพที่ไม่มีในขั้นตอนใดและไม่มีหน่วยงานอิสระเคยตรวจสอบ

ข้อสามคือประเด็นของภาคนี้ ผู้ควบคุมไม่ได้อยู่นอกส่วนเผื่อที่คำนวณแล้วพบว่าพอ พวกเขาอยู่นอกบริเวณที่ไม่มีใครเคยคำนวณส่วนเผื่อเลย และเครื่องมือวัดในห้องควบคุมไม่มีทางบอกพวกเขาเรื่องนั้นได้

สิ่งที่กรณีนี้สอน: ก่อนรันการทดสอบที่ต้องปิดระบบป้องกัน ให้เขียนก่อนว่าสภาพที่มันจะพาระบบไปนั้นมีใครวิเคราะห์ไว้แล้วหรือยัง — ถ้ายัง การทดสอบคืออุบัติเหตุที่นัดเวลาไว้

Therac-25 · 1985–1987

เครื่องฉายรังสีรักษาเครื่องหนึ่งทำสองอย่างด้วยลำอิเล็กตรอนลำเดียวกัน ยิงลำกำลังต่ำเข้าตัวคนไข้โดยตรง หรือยิงลำกำลังสูงเข้าใส่ target แล้วให้ target เปลี่ยนมันเป็น X-ray ความต่างระหว่างการรักษากับการฆ่า อยู่ที่ชิ้นโลหะชิ้นหนึ่งอยู่หรือไม่อยู่ในแนวลำ

ระหว่างวันที่ 3 มิถุนายน 1985 ถึง 17 มกราคม 1987 เครื่องฉายรังสี Therac-25 ให้รังสีเกินขนาดกับคนไข้หกรายในโรงพยาบาลสี่แห่งใน Georgia, Ontario, Washington และ Texas อย่างน้อยสามคนตาย — คนไข้ที่ Tyler รัฐ Texas ทั้งสองและคนไข้ Yakima รายสุดท้ายที่ตายเดือนเมษายน 1987 บางแหล่งนับสี่ เล่มนี้ยืนยันรายที่สี่ไม่ได้ เดือนกุมภาพันธ์ 1987 FDA สั่งเรียกคืนระดับ Class I

ส่วนเผื่อของเครื่องรุ่นก่อนถูกถอดออกทั้งชุด และเหตุผลที่ถอดคือเหตุผลที่ฟังขึ้นที่สุดในเรื่องนี้ Therac-6 และ Therac-20 มีประวัติใช้งานสะอาด และพึ่งฮาร์ดแวร์เป็นหลักในการคุมความปลอดภัย interlock ทางกายภาพทำให้จ่ายลำอิเล็กตรอนกำลังสูงไม่ได้ถ้า target กับ collimator ไม่อยู่ในแนวลำ Therac-25 ย้ายการตรวจสอบเหล่านั้นไปไว้ในซอฟต์แวร์และถอด interlock ฮาร์ดแวร์ออก ซอฟต์แวร์ที่ย้ายไปนั้นสืบทอดมาจากเครื่องที่ไม่เคยทำร้ายใคร จึงถือว่าพิสูจน์แล้ว

มันไม่ได้ถูกพิสูจน์ มันถูกบัง ข้อบกพร่องเดียวกันอยู่ในโค้ดของ Therac-20 มาตลอด ไม่มีใครรู้เพราะ interlock ของ Therac-20 จับสภาวะที่ผิดไว้ได้ทุกครั้ง interlock ไม่ได้ซ้ำซ้อนกับซอฟต์แวร์ มันคือสิ่งเดียวที่ความจริงเคยทดสอบ ถอดมันออกคือถอดส่วนเผื่อทั้งหมดที่เคยพิสูจน์ได้ เหลือไว้แต่คำอ้าง

อุบัติเหตุที่ Tyler ทั้งสองครั้งเกิดจาก Malfunction 54 race condition ในโค้ดควบคุมบน PDP-11 ถ้าผู้ปฏิบัติงานป้อนใบสั่งฉาย แล้วเลื่อนเคอร์เซอร์ขึ้นไปแก้โหมดลำจาก X-ray เป็นอิเล็กตรอนเร็วพอ งานป้อนข้อมูลกับงานตั้งค่าเครื่องจะเห็นสภาวะเครื่องไม่ตรงกัน เครื่องจึงยิงลำอิเล็กตรอนในโหมด X-ray กำลังสูงโดยไม่มี target อยู่ในแนวกระแสเต็มลงบนตัวคนไข้

หน้าต่างเวลานั้นแคบพอที่มีเพียงผู้ปฏิบัติงานที่คล่องเท่านั้นเข้าถึงได้ ตัวเลขแปดวินาทีที่ยกกันทั่วไป เล่มนี้ยังไม่ได้ตรวจกับต้นฉบับของ Leveson ข้อที่ควรอ่านช้าๆ

คือข้อนี้ ทุกอุบัติเหตุมีความชำนาญนำหน้า ไม่ใช่ความสะเพร่า คนที่พิมพ์เร็วกว่าคือคนที่ไปถึงสภาวะที่ไม่มีใครเคยนับ

ฉบับที่เล่ากันว่าบั๊กตัวหนึ่งฆ่าคน ผิดที่คำว่าตัวหนึ่ง Malfunction 54 ทำให้เกิดอุบัติเหตุที่ Tyler ส่วน Hamilton และ Yakima ทั้งสองครั้งเกิดจากข้อบกพร่องคนละตัว Nancy Leveson และ Clark Turner เขียนใน IEEE Computer เดือนกรกฎาคม 1993 ว่าการไล่จับบั๊กทีละตัวไม่ใช่ทางสร้างระบบที่ปลอดภัย ความปลอดภัยของซอฟต์แวร์เป็นคุณสมบัติของระบบ ไม่ใช่ของซอฟต์แวร์ AECL หาสาเหตุหนึ่ง แก้ แล้วประกาศว่าเครื่องปลอดภัย หลังอุบัติเหตุทุกครั้ง

ส่วนเผื่อถูกคิดกับประวัติสะอาดของเครื่องรุ่นก่อน สิ่งที่โจมตีคือสภาวะเครื่องที่ไม่มีใครเคยนับ เข้าถึงได้ผ่านหน้าต่างเวลาระหว่างสองงานที่รันพร้อมกันกับมนุษย์ที่เร็ว และประวัติสะอาดนั้นก็หมุนเป็นวงกลม ส่วนเผื่อถูกถอดเพราะสิ่งที่มันกันไม่เคยถูกเห็น และมันไม่เคยถูกเห็นเพราะส่วนเผื่ออยู่ตรงนั้น

สิ่งที่กรณีนี้สอน: ก่อนถอดตัวป้องกันที่ไม่เคยทำงาน ให้ถามว่าประวัติสะอาดนั้นเป็นหลักฐานของโค้ดหรือของตัวป้องกัน — แล้วรันโค้ดโดยไม่มีมันบนโต๊ะทดสอบก่อนบนตัวคนไข้

ฟุกุชิมะ ไดอิจิ · 11 มีนาคม 2011

โรงไฟฟ้าบนชายฝั่งทุกแห่งต้องตอบคำถามเดียวกัน น้ำจะขึ้นมาได้สูงสุดเท่าไร คำตอบมาจากบันทึกที่มีอยู่ และบันทึกก็มีอยู่เท่าที่มันมี ปี 2002 TEPCO ประเมินสึนามิใหม่โดยสมัครใจด้วยวิธีของ Japan Society of Civil Engineers แล้วตั้งค่าออกแบบไว้ที่ 5.7 เมตร

วันที่ 11 มีนาคม 2011 สึนามิสูงราว 15 เมตรมาถึงโรงไฟฟ้า Fukushima Daiichi โรงไฟฟ้าที่ออกแบบรับสึนามิ 5.7 เมตร บนพื้นที่สูงจากระดับน้ำทะเลราว 10 เมตร

ตัวเลขที่มาถึงในปี 2011 TEPCO คำนวณได้เองในปี 2008

ไม่มีใครตายจากรังสีเฉียบพลัน ปี 2018 รัฐบาลญี่ปุ่นรับรองว่าคนงานอายุห้าสิบกว่าที่ตายด้วยมะเร็งปอดหลังรับรังสีราว 195 mSv ตายจากรังสี เพื่อจ่ายค่าชดเชย และคำ

รับรองนั้นถูกโต้แย้งบนหลักระบาดวิทยา คนที่ตายจำนวนมากคือจากการอพยพ ราว 2,200 คน ตัวเลขต่างกันตามวิธีนับ

ส่วนเผื่อมีสองชั้น ชั้นแรกคือค่าออกแบบ 5.7 เมตร ชั้นที่สองคือความสูงของพื้นดินเอง พื้นหลักของหน่วย 1–4 อยู่สูงจากระดับน้ำทะเลราว 10 เมตร ถ้าน้ำข้ามชั้นแรกได้ก็ยังมีอีกสี่เมตรกว่ารออยู่ ใต้พื้นนั้นคือชั้นใต้ดินและอาคารต่ำที่วางเครื่องกำเนิดไฟฟ้าดีเซลฉุกเฉินกับสวิตช์เกียร์ไว้

น้ำที่มาถึงสูงสุด 15.7 เมตรที่ปลายด้านใต้ของพื้นที่ตามการประมาณ ตรงกับระดับน้ำท่วม 15.5 เมตรที่วัดได้ที่โรงไฟฟ้า ฝั่งภูเขาน้ำท่วมถึง O.P. +14 ถึง +15 เมตร คลื่นใหญ่สุดราว 13–14 เมตร ข้ามพื้น 10 เมตรไป

น้ำท่วมชั้นใต้ดิน ทำลายเครื่องดีเซลกับสวิตช์เกียร์ DC และโรงไฟฟ้าเข้าสู่ station blackout ส่วนเผื่อทั้งสองชั้นถูกกินหมดด้วยเหตุการณ์เดียว สองชั้นที่นับแยกกันบนกระดาษ เป็นชั้นเดียวกันในน้ำ นี่คือความล้มเหลวตามนิยามของ defence-in-depth ที่ชั้นป้องกันสัมพันธ์กัน

เรื่องปี 2008 เป็นแบบนี้ ระหว่างทบทวนความปลอดภัยด้านแผ่นดินไหว TEPCO ทดลองใช้แบบจำลองแหล่งกำเนิดคลื่นของแผ่นดินไหว Meiji Sanriku ปี 1896 กับร่องลึกนอกชายฝั่ง Fukushima ผลคือ 15.7 เมตร บริษัทตั้งคำถามกับแบบจำลอง ขอให้คำนวณด้วยวิธีอื่น ได้ผลเกือบเท่าเดิม แล้ววางผลนั้นไว้เพื่อสั่งศึกษาทางวิศวกรรมเพิ่ม

ผลลัพธ์ไม่ได้หาย ไม่ได้ถูกเข้าใจผิด มันถูกจัดประเภทใหม่จากข้อกำหนดทางวิศวกรรมเป็นคำถามวิจัย ข้อกำหนดมีวันครบกำหนด คำถามวิจัยไม่มี

ฉบับที่ว่านี่คือภัยธรรมชาติที่คาดไม่ถึง ถูกคณะกรรมการอิสระของรัฐสภาญี่ปุ่นเองปฏิเสธ NAIIC รายงานวันที่ 5 กรกฎาคม 2012 ว่าอุบัติเหตุนี้ไม่อาจถือเป็นภัยธรรมชาติ แต่เป็น "profoundly manmade disaster" ที่คาดการณ์และป้องกันได้และควรทำ คณะกรรมการใช้คำว่า "collusion" ระหว่างรัฐบาล ผู้กำกับดูแล และอุตสาหกรรมนิวเคลียร์ และเขียนว่าความเสี่ยงสึนามิถูกเพิกเฉยทั้งหมด

ส่วนเผื่อถูกคิดจากหน้าต่างบันทึกประวัติศาสตร์ที่ไม่มีเหตุการณ์นี้อยู่ แล้วแปลงเป็นความสูงตัวเดียว แต่ความล้มเหลวที่ลึกกว่านั้นไม่ได้อยู่ที่ความสูง มันอยู่ที่ว่าองค์กรที่คำนวณส่วนเผื่อ เป็นคนตัดสินเองว่าจะเชื่อตัวเลขที่ตัวเองคำนวณซ้ำหรือไม่

สิ่งที่กรณีนี้สอน: เมื่อการคำนวณซ้ำให้ตัวเลขที่ทำลายค่าออกแบบเดิม ให้ส่งมันไปคนที่ไม่ได้เป็นเจ้าของค่าออกแบบ — อย่าให้ผู้เผื่อเป็นผู้ตัดสินว่าจะเชื่อส่วนเผื่อใหม่หรือไม่

โบอิ้ง 737 MAX · 2018–2019

เครื่องยนต์ LEAP-1B ใหญ่กว่าของเดิมและถูกเลื่อนไปข้างหน้า ผลที่ตามมาไม่ได้อยู่ที่เครื่องยนต์ ของที่เปลี่ยนไปคือโมเมนต์ก้มเงยที่ angle of attack สูง แรงคันบังคับที่การรับรองต้องการจึงไม่เป็นไปตามนั้นอีก ทางแก้ของ Boeing ชื่อ MCAS มันสั่ง stabiliser ปรับหัวลงเพื่อดันแรงคันบังคับกลับเข้ารูป มันไม่ใช่ระบบกัน stall

วันที่ 29 ตุลาคม 2018 Lion Air เที่ยวบิน 610 ทะเบียน PK-LQP ตกลงทะเลชวาหลังออกจาก Jakarta ไม่นาน ตาย 189 คน วันที่ 10 มีนาคม 2019 Ethiopian Airlines เที่ยวบิน 302 ตกหลังออกจาก Addis Ababa ตาย 157 คน ภายในไม่กี่วันเครื่อง MAX ทั้งโลกถูกสั่งจอด นาน 20 เดือน

Boeing ไม่ใส่ MCAS ในคู่มือปฏิบัติการของนักบิน ลูกเรือ Lion Air จึงไม่มีชื่อเรียกสิ่งที่กำลังสู้กับพวกเขา

ส่วนเผื่อที่ยื่นให้ FAA คือ 0.6 องศา การวิเคราะห์ความปลอดภัยของ Boeing ประเมิน MCAS เป็นฟังก์ชันอำนาจต่ำ ที่ความเร็วต่ำมันสั่ง stabiliser หัวลงได้ 0.6 องศาต่อครั้ง และบนตัวเลขนั้น การประเมินก็ถูก 0.6 องศาบน stabiliser ที่ trim ได้คือความรำคาญที่นักบิน trim ออกได้

ระหว่างทดสอบบิน วิศวกรเพิ่มอำนาจความเร็วต่ำเป็น 2.5 องศาต่อครั้ง โดยไม่ทบทวนการวิเคราะห์ความปลอดภัยกับตัวเลขใหม่ และครั้งเดียวไม่ใช่ขีดจำกัด MCAS รีเซ็ตทุกครั้งที่นักบิน trim ด้วยไฟฟ้า แล้วสั่งหัวลงใหม่ราวห้าวินาทีถัดมา ไม่จำกัดระยะสะสม ทุกครั้งที่นักบินแก้ เขาก็ให้สิทธิ์มันสั่งใหม่ไปด้วยพร้อมกัน

รายงานข่าวจากรายงานเบื้องต้นของ Lion Air 610 ระบุมากกว่า 24 ครั้ง ตัวเลขนี้เล่มนี้ยังไม่ได้ตรวจกับรายงานฉบับสุดท้ายของ KNKT

อำนาจจริงจึงไม่ใช่ 0.6 ไม่ใช่ 2.5 แต่คือ 2.5 องศาซ้ำไม่จำกัด เท่ากับระยะหัวลงเต็มของ stabiliser ต่อลูกเรือที่แรง elevator หมดก่อน ตัวเลขที่ผ่านประตูรับรองกับตัวเลขที่ขึ้นไปบินจริง เป็นคนละตัว และไม่มีใครเอาสองตัวนั้นมาวางข้างกัน

MCAS อ่านจากเซนเซอร์ angle of attack ตัวเดียว ไม่เทียบ ไม่โหวต ของ Lion Air เพิ่งถูกเปลี่ยนเมื่อวันก่อนด้วยชิ้นที่ปรับเทียบผิด แบบที่แก้หลังจอดบอกรูปร่างของแบบเดิมได้ชัดที่สุด ตอนนี้ MCAS เทียบเซนเซอร์สองตัว ไม่ทำงานถ้าต่างกัน 5.5 องศาขึ้นไปตอนเก็บ flap รีเซ็ตไม่ได้ และรวมทั้งหมดไม่เกิน 2.5 องศา ทุกข้อในรายการนั้นคือชื่อของสิ่งที่ขาดไปก่อนหน้า

ฉบับที่ว่าสัญญาณเตือน AOA DISAGREE ล้มเหลว ไม่ถูก มันถูกผูกไว้กับตัวแสดง AOA ที่เป็นตัวเลือกเสริม จึงทำงานเฉพาะสายการบินที่ซื้อตัวเลือกนั้น ทั้งที่ข้อกำหนดออกแบบของ Boeing เองให้มันเป็นมาตรฐาน วิศวกร Boeing พบเรื่องนี้ในปี 2017 ตัดสินว่าไม่จำเป็นต่อความปลอดภัย และไม่บอกทั้งสายการบินและ FAA จนหลัง Lion Air ตก

ฉบับที่ว่านักบินตะวันตกจะเอาอยู่ ขัดกับบันทึกการรับรอง JATR รายงานต่อ FAA เดือนตุลาคม 2019 ว่า MCAS "was not evaluated as a complete and integrated function" ในเอกสารที่ Boeing ยื่น การเปลี่ยนแปลงถูกหั่นเป็นชิ้น แต่ละชิ้นผ่านประตูของตัวเอง ไม่มีใครดูทั้งก้อน

สิ่งที่กรณีนี้สอน: ทุกครั้งที่ตัวเลขอำนาจของฟังก์ชันเปลี่ยน ให้รันการวิเคราะห์ความปลอดภัยใหม่บนตัวเลขที่ส่งไปเครื่องจริง — และคูณด้วยจำนวนครั้งที่มันรีเซ็ตได้

ท่าที่เอาไปใช้ได้: เขียนชื่อของแรงที่ค่าความปลอดภัยของคุณหารอยู่ให้เป็นคำนามหนึ่งคำ แล้วถามว่าสิ่งที่จะฆ่าระบบนี้เป็นคำนามเดียวกันหรือไม่ ถ้าไม่ใช่ ตัวเลขนั้นไม่ได้เผื่ออะไรเลย

จะพลิกคำตัดสินของภาคนี้ได้เมื่อ: ผู้อ่านชี้ได้ว่ารายงานสอบสวนหลักของกรณีใดในเก้ากรณีนี้สรุปว่าแรงที่ทำลายโครงสร้างคือแรงชนิดเดียวกับที่แบบตั้งชื่อไว้ และมีขนาดเกินค่าที่เผื่อ กรณีนั้นออกจากภาคนี้ทันที และถ้าเกินสามกรณี ชื่อภาคนี้ผิด

แหล่งของภาคนี้:

- ทาโคมาแนโรวส์ — ประวัติทางการของ WSDOT · https://wsdot.wa.gov/tnbhistory/bridges-failure.htm
- ทาโคมาแนโรวส์ — Billah & Scanlan, American Journal of Physics 59(2), 1991 · https://pubs.aip.org/aapt/ajp/article/59/2/118/1053909/Resonance-Tacoma-Narrows-bridge-failure-and
- ทาโคมาแนโรวส์ — Billah & Scanlan, PDF · https://www.aapt.org/store/upload/tacoma_narrows2.pdf
- คอเม็ต — Aerossurance, Comet misconceptions · https://aerossurance.com/safety-management/comet-misconceptions/
- คอเม็ต — FAA Lessons Learned, Comet 1 · https://lessonslearned.faa.gov/ll_main.cfm?TabID=1&LLID=28&LLTypeID=2
- คอเม็ต — รายงานศาลสอบสวน CAP 127 · http://lessonslearned.faa.gov/Comet1/G-ALYP_Report.pdf
- คอเม็ต — รายงานอุบัติเหตุ G-ALYY · https://asn.flightsafety.org/reports/1954/19540408_COMT_G-ALYY.pdf
- ไททานิก — รายงาน Wreck Commissioner ส่วน Board of Trade's Administration · https://www.titanicinquiry.org/BOTInq/BOTReport/botRepBOT.php
- ไททานิก — คำให้การของ Chalmers · https://www.titanicinquiry.org/BOTInq/BOTInq23Chalmers01.php
- ไททานิก — คำให้การของ Archer · https://www.titanicinquiry.org/BOTInq/BOTInq25Archer01.php
- ไททานิก — McLean, "Regulation run mad" · https://www.nuff.ox.ac.uk/Users/McLean/Titpaper.PDF
- Hyatt Regency — NIST · https://www.nist.gov/el/walkway-collapse-kansas-city-missouri-1981
- Hyatt Regency — NBSIR 82-2465 · https://nvlpubs.nist.gov/nistpubs/Legacy/IR/nbsir82-2465.pdf
- Hyatt Regency — สรุปข้อสรุปของ NBS, UPI 26 กุมภาพันธ์ 1982 · https://www.upi.com/Archives/1982/02/26/Here-is-a-summary-of-conclusions-by-the-National/6635097243860/
- Hyatt Regency — การเพิกถอนใบอนุญาต, UPI 1986 · https://www.upi.com/Archives/1986/01/22/The-two-engineers-who-designed-the-Hyatt-Regency-Hotel/5084506754000/

- แชลเลนเจอร์ — รายงาน Rogers บทที่ IV · https://www.nasa.gov/history/rogersrep/v1ch4.htm
- แชลเลนเจอร์ — รายงาน Rogers บทที่ V · https://www.nasa.gov/history/rogersrep/v1ch5.htm
- แชลเลนเจอร์ — Feynman, ภาคผนวก F · https://calteches.library.caltech.edu/3570/1/Feynman.pdf
- เชอร์โนบิล — IAEA INSAG-7 · https://www-pub.iaea.org/MTCD/publications/PDF/Pub913e_web.pdf
- เชอร์โนบิล — World Nuclear Association · https://world-nuclear.org/information-library/safety-and-security/safety-of-plants/chernobyl-accident
- เชอร์โนบิล — ลำดับเหตุการณ์ · https://world-nuclear.org/information-library/appendices/chernobyl-accident-appendix-1-sequence-of-events
- เชอร์โนบิล — UNSCEAR · https://www.unscear.org/unscear/en/chernobyl.html
- Therac-25 — Leveson & Turner, IEEE Computer, กรกฎาคม 1993 · https://www.cs.columbia.edu/~junfeng/08fa-e6998/sched/readings/therac25.pdf
- Therac-25 — Leveson, ส่วนที่ V บทเรียน · https://cse.msu.edu/~cse470/Public/Handouts/Therac/Therac_5.html
- Therac-25 — บันทึกอุบัติเหตุ Tyler 1986 · https://www.johnstonsarchive.net/nuclear/radevents/1986USA1.html
- ฟุกุชิมะ — รายงาน NAIIC · https://www.nirs.org/wp-content/uploads/fukushima/naiic_report.pdf
- ฟุกุชิมะ — TEPCO ว่าด้วยการคำนวณ 15.7 เมตรปี 2008 · https://www.tepco.co.jp/en/nu/fukushima-np/info/12042401-e
- ฟุกุชิมะ — Acton & Hibbs, Carnegie · https://carnegieendowment.org/2012/03/06/why-fukushima-was-preventable-pub-47361
- ฟุกุชิมะ — รายงานรัฐบาลญี่ปุ่นต่อ IAEA, ความเสียหายจากแผ่นดินไหวและสึนามิ · https://japan.kantei.go.jp/kan/topics/201106/pdf/chapter_iii-2.pdf
- ฟุกุชิมะ — NPR, การรับรองการตายจากรังสีรายแรก 2018 · https://www.npr.org/2018/09/05/644933882/japanese-government-acknowledges-first-fukushima-radiation-death
- 737 MAX — รายงาน JATR ตุลาคม 2019 · https://www.faa.gov/sites/faa.gov/files/2021-08/Final_JATR_Submittal_to_FAA_Oct_2019.pdf
- 737 MAX — รายงานฉบับสุดท้าย KNKT, PK-LQP · https://www.aaiu.ie/sites/default/files/FRA/2018%20-%20035%20-%20PK-LQP%20Final%20Report.pdf
- 737 MAX — Seattle Times, การเพิ่มอำนาจ MCAS · https://www.seattletimes.com/seattle-news/times-watchdog/the-inside-story-of-mcas-how-boeings-737-max-system-gained-power-and-lost-safeguards/
- 737 MAX — แถลงการณ์ Boeing เรื่อง AOA Disagree, พฤษภาคม 2019 · https://boeing.mediaroom.com/news-releases-statements?item=130431

• 737 MAX — Boeing, MCAS หลังแก้ · https://www.boeing.com/737-max-updates/mcas/

3 · ศัตรูที่อ่านแบบแปลนของคุณ

ลมไม่เคยจำได้ว่าครั้งที่แล้วมันพัดไม่ล้ม พายุลูกหน้ามาด้วยฟิสิกส์เดิม ไม่สนใจว่าคุณไปเสริมเสาต้นไหนมาหลังจากครั้งก่อน ความล้า ความดัน คลื่น ก็เหมือนกันหมด ไม่มีตัวไหนเคยเปิดแบบก่อสร้างของคุณดูว่าตรงไหนบางที่สุด ส่วนเผื่อจนถึงหน้านี้จึงทำงานได้ด้วยวิธีตรงไปตรงมา คุณเดาแรงที่แย่ที่สุดที่ธรรมชาติจะออกมาได้ แล้วเผื่อไว้เกินนั้น

ตั้งแต่หน้านี้ไปอีกฝ่ายเปิดแบบดู เขาอ่านแบบของคุณ อ่านคู่มือปฏิบัติงานของคุณ อ่านวิธีที่คุณใช้ตรวจหาเขา แล้วเลือกจุดที่จะดัน เสาที่คุณเสริมแล้วเขาเดินข้าม เขาไปที่ต้นที่คุณยังไม่ได้เสริม และถ้าคุณเสริมทุกต้นเท่ากันหมด เขาก็เปลี่ยนไปที่คนเดิน เครื่องแทน

ภาคนี้คือบานพับของเล่ม เพราะส่วนเผื่อเปลี่ยนความหมายทั้งหมดตรงนี้ ในโลกก่อนหน้านี้มันคือกันชน ในโลกนี้มันคือตัวเลขที่ถูกวัดแล้วเดินอ้อม แปดกรณีในภาคนี้กินเวลาตั้งแต่ปี 1851 ถึง 1985 และตอบคำถามเดียวกันว่า เมื่ออีกฝ่ายถือแบบแปลนฉบับเต็มอยู่ในมือ อะไรยังป้องกันได้

หลักการของ Kerckhoffs · 1883

รหัสทางทหารก่อนหน้านั้นถูกออกแบบบนความหวังข้อเดียว คือวิธีของมันจะไม่หลุดไปถึงอีกฝ่าย แต่วิธีเข้ารหัสเป็นของที่ต้องแจกออกไปให้คนใช้ ต้องเดินทางไปกับหน่วยที่เคลื่อนที่ ต้องอยู่ในมือคนที่ถูกจับได้ ความมั่นคงที่วางบนความหวังแบบนั้นมีอายุเท่ากับความลับที่เปราะที่สุดในกองทัพ และในปี 1883 มีคนเขียนลงกระดาษเป็นครั้งแรกว่าอย่าวางมันไว้ตรงนั้น

เดือนมกราคม 1883 Journal des sciences militaires เล่มที่ IX พิมพ์ตอนแรกของบทความ La cryptographie militaire ของ Auguste Kerckhoffs ที่หน้า 5 ถึง 38 ตอนที่สองตามมาเดือนกุมภาพันธ์ที่หน้า 161 ถึง 191 ตอนแรกวางข้อกำหนดหกข้อสำหรับระบบรหัสทางทหาร และข้อที่สองในหกข้อนั้นคือสิ่งที่โลกปัจจุบันเรียกว่าหลักการของ Kerckhoffs

ข้อที่สองบอกว่าระบบต้องไม่ต้องการความลับ และต้องตกไปอยู่ในมือศัตรูได้โดยไม่เดือดร้อน ต้นฉบับภาษาฝรั่งเศสคือ « Il faut qu'il n'exige pas le secret, et qu'il puisse sans inconvénient tomber entre les mains de l'ennemi » สังเกตให้ดีว่าเขาไม่ได้ขอให้ระบบแข็งแรงขึ้น เขาขอให้ย้ายเส้นแบ่ง จนของที่ต้องปิดเหลือชิ้นเดียว แบบถูกถือว่าอยู่ในมืออีกฝ่ายตั้งแต่วันที่ออกแบบ สิ่งเดียวที่ยังปิดได้คือกุญแจ นี่คือคำประกาศเป็นทางการครั้งแรกว่าความลับของแบบไม่ใช่ความมั่นคง

ฉบับที่เล่ากันว่า Kerckhoffs สั่งให้เปิดเผยอัลกอริทึม ไม่ตรงกับข้อความ เขาบอกว่าระบบต้องไม่พึ่งความลับ ไม่ได้บอกว่าต้องพิมพ์ และระยะห่างระหว่างสองอย่างนั้นคือทั้งหมดที่เรื่องนี้ต่างกัน Fabien Petitcolas ผู้ขอวารสารฉบับนี้จาก British Library มาพิมพ์ข้อความเต็มในปี 1998 สรุปว่าหลักการนี้ให้ถือว่าวิธีเป็นสาธารณะระหว่างออกแบบเท่านั้น การพิมพ์เป็นกลยุทธ์ ตัวหลักการเป็นข้อสมมติเรื่องศัตรู

ฉบับที่ว่าเขาเรียกร้องระบบที่เจาะไม่ได้ทางคณิตศาสตร์ กลับด้านกับต้นฉบับ ข้อที่หนึ่งใช้คำว่า matériellement, sinon mathématiquement — เจาะไม่ได้ในทางปฏิบัติ ถ้าไม่ได้ในทางคณิตศาสตร์ เขาเขียนต่อทันทีว่าทุกคนยอมรับสามข้อหลัง ไม่มีใครยอมรับสามข้อแรก แล้วตอบคำถามของตัวเองว่าระบบที่เจาะไม่ได้ทางคณิตศาสตร์และผ่านข้อสองไปพร้อมกัน เป็นสิ่งที่เป็นไปไม่ได้ทางคณิตศาสตร์ ส่วนเผื่อในปี 1883 จึงเป็นส่วนเผื่อทางปฏิบัติตั้งแต่บรรทัดแรก

และครึ่งหนึ่งของหกข้อไม่ใช่กฎเรื่องความลับเลย กุญแจต้องจำได้โดยไม่จด ระบบต้องใช้กับโทรเลขได้ พกพาได้ คนเดียวใช้ได้ ไม่ต้องเครียด ไม่ต้องจำกฎยาว ครึ่งนั้นเขียนถึงคนของตัวเอง ไม่ได้เขียนถึงศัตรู แบบจำลองความมั่นคงของ Kerckhoffs จึงมีผู้ปฏิบัติงานยืนอยู่ในนั้นตั้งแต่ปี 1883 เพราะระบบที่คนใช้ไม่ไหว คือระบบที่จะถูกใช้ผิด

สิ่งที่กรณีนี้สอน: เขียนรายการทุกสิ่งที่ระบบของคุณต้องปิดเป็นความลับ แล้วขีดฆ่าทุกบรรทัดที่ไม่ใช่กุญแจ — ถ้ายังป้องกันได้ ระบบผ่านข้อสอง ถ้าไม่ได้ ส่วนเผื่อของคุณมีอายุแค่จนกว่าจะมีใครอ่าน

ข้อโต้แย้งเรื่องกุญแจ · ลอนดอน 1851

ที่หน้าร้านเลขที่ 124 Piccadilly มีกุญแจดอกหนึ่งตั้งโชว์อยู่ตั้งแต่ปี 1790 พร้อมป้ายสัญญาว่าใครทำเครื่องมือเปิดมันได้จะได้รับ 200 กินีทันที คนที่เดินผ่านถนนเส้นนั้นเห็นป้ายเดิมอยู่ตรงนั้นมา 61 ปีโดยไม่มีใครมารับเงินไป และนั่นคือหลักฐานทั้งหมดที่อังกฤษมีว่ากุญแจของตัวเองแน่นหนา ไม่ใช่บทพิสูจน์ แต่เป็นคำท้าที่ว่างอยู่นานพอจนคนเลิกสงสัย

ปี 1851 ระหว่างงาน Great Exhibition ที่ลอนดอน Alfred Charles Hobbs ช่างกุญแจจาก Boston รับคำท้านั้น เขาใช้เวลาราว 51 ชั่วโมงกระจายใน 16 วัน แล้วไขกุญแจท้าทายของ Bramah ดอกนั้นได้ ตัวเลข 61 ปีมาจากการนับปี แหล่งหนึ่งนับว่าเกิน 67 เล่มนี้ใช้เลขปี เงื่อนไขตกลงกันไว้ล่วงหน้า ทำที่ร้าน Bramah ใช้เครื่องมืออะไรก็ได้ กุญแจต้องไม่เสียหาย คณะกรรมการโต้แย้งเรื่องสภาพการทดสอบ เถียงกันจบแล้ว 200 กินีถูกจ่าย และ Hobbs วางเหรียญโชว์ในงานจนจบ เขาไขกุญแจ detector หกคานของ Chubb รุ่นสิทธิบัตรปี 1818 ได้ด้วยเป็นคนแรก วันและเวลาที่ไขมีเฉพาะในเรื่องเล่า เล่มนี้ไม่พิมพ์

ฉบับที่เล่ากันว่าเรื่องนี้ทำลายอุตสาหกรรมกุญแจอังกฤษ ไม่ตรงกับสิ่งที่เกิดขึ้นหลังงาน ยอดขายของ Bramah กับ Chubb ไม่เสียหาย ด้วยเหตุผลที่ตรงไปตรงมา กุญแจที่ทนผู้เชี่ยวชาญได้ 51 ชั่วโมงยังเป็นกุญแจดี บริษัทที่ตายจริงคือนายจ้างของ Hobbs เอง Day & Newell แห่งนิวยอร์ก กุญแจ parautoptic ที่เขานำมาโชว์ถูก Linus Yale ไขได้ทันทีหลังงาน

ของที่เหลือจากปีนั้นมาถึงวันนี้จึงไม่ใช่กุญแจดอกไหน แต่คือคำถามที่การไขเปิดขึ้นมา ว่าควรถกเรื่องความไม่ปลอดภัยของกุญแจในที่เปิดเผยหรือไม่ คำตอบอยู่ในหนังสือ Rudimentary Treatise on the Construction of Locks พิมพ์ที่ลอนดอนปี 1853 บทที่หนึ่ง และคำตอบคือควร โจรรู้เรื่องไขกุญแจมากกว่าที่ใครจะสอนได้อยู่แล้ว ถ้ากุญแจดอกไหนไม่แน่นหนาเท่าที่เชื่อกัน คนสุจริตต่างหากที่ต้องรู้ เพราะคนทุจริตแทบแน่นอนว่าจะเป็นฝ่ายแรกที่เอาความรู้นั้นไปใช้

ตัวอย่างที่หนังสือยกไม่ใช่กุญแจ แต่คือนมปลอมปนในลอนดอน คนขายนมรู้วิธีปลอมปนอยู่แล้ว การเปิดโปงไม่ได้สอนอะไรเขาเลยสักอย่าง มันสอนคนซื้อให้ตรวจ นี่คือข้อโต้แย้งเรื่องข้อมูลผู้บริโภค ไม่ใช่จริยธรรมของนักเจาะ

คนเขียนย่อหน้านั้นไม่ใช่ Hobbs คำนำลงชื่อ C. Tomlinson กรกฎาคม 1853 และเล่าลำดับไว้ครบ Tomlinson ปรึกษาศาสตราจารย์ Cowper ในเดือนสิงหาคม 1852 ว่าควรอธิบายข้อบกพร่องของกุญแจให้คนทั่วไปหรือไม่ ก่อนที่ Hobbs จะเข้ามาเกี่ยว Cowper แนะนำให้เขารู้จัก Hobbs ผู้ส่งวัตถุดิบมา George Dodd เรียบเรียง Tomlinson แก้และเพิ่ม Hobbs อ่านปรู๊ฟและใส่หมายเหตุ Wikipedia ใส่ประโยคนี้ในปากของ Hobbs ถ้าต้องมีชื่อเดียว ชื่อนั้นคือ Tomlinson

สิ่งที่กรณีนี้สอน: เมื่อมีคนรายงานช่องโหว่ ให้เขียนชื่อคนที่ได้ประโยชน์จากการไม่พูดแล้วเทียบกับชื่อคนที่ได้ประโยชน์จากการรู้ — ถ้าฝั่งแรกมีแค่คนขาย ให้พิมพ์

le chiffre indéchiffrable · 1854–1863

รหัสตัวหนึ่งเดินทางมาพร้อมชื่อเล่นที่เป็นคำรับประกันไปในตัว le chiffre indéchiffrable รหัสที่ถอดไม่ได้ คนที่ใช้มันฝากความลับของตัวเองไว้กับคำสามคำนั้น และคำสามคำนั้นเป็นจริงอยู่ตราบเท่าที่ยังไม่มีใครพิมพ์วิธีถอดออกมา ไม่ใช่ตราบเท่าที่ยังไม่มีใครถอดได้ ระยะห่างระหว่างสองเงื่อนไขนั้นคือเนื้อเรื่องทั้งหมดของกรณีนี้

ปี 1863 Friedrich W. Kasiski พิมพ์ Die Geheimschriften und die Dechiffrir-Kunst ที่เบอร์ลิน หนา 95 หน้า และเป็นครั้งแรกที่มีวิธีทั่วไปตีพิมพ์สำหรับเจาะรหัสหลายตัวอักษรที่ใช้คำสำคัญซ้ำ

วิธีนั้นอ่านแบบของรหัสตรงจุดที่มันซ้ำตัวเอง คำเดิมที่บังเอิญตกลงตรงตำแหน่งเดิมของกุญแจจะออกมาเป็นตัวอักษรชุดเดิมทุกครั้ง ระยะห่างระหว่างการซ้ำสองครั้งจึงเป็นทวีคูณของความยาวกุญแจเสมอ ที่เหลือเป็นเลขคณิต หาชุดตัวอักษรตั้งแต่สามตัวที่ปรากฏซ้ำในรหัส วัดระยะห่าง แยกตัวประกอบ ตัวประกอบร่วมคือความยาวกุญแจ รู้ความยาวแล้วก็แยกข้อความเป็นคอลัมน์ แล้วโจมตีแต่ละคอลัมน์ด้วยความถี่ตัวอักษร วิธีปรับปรุงสมัยใหม่คือ index of coincidence

Charles Babbage ทำได้ก่อน และไม่พิมพ์อะไรเลย ปีที่เขาทำได้ แหล่งสองแหล่งไม่ตรงกัน และเล่มนี้พิมพ์ทั้งสอง Franksen ผู้อ่านสมุดบันทึกของ Babbage ให้ว่าเร็วถึงปี 1846 ส่วนปี 1854 มาจากเหตุการณ์ที่ John Hall Brock Thwaites ทันตแพทย์จาก Bristol ส่งรหัส "ใหม่" ไป Journal of the Society of Arts Babbage ชี้ว่ามันคือ Vigenère Thwaites ท้าให้เจาะ และ Babbage เจาะได้ Simon Singh ใน The Code Book หน้า 78 อยู่หลังปีนั้น ทั้งสองเป็นจริงพร้อมกันได้ วิธีอยู่ในสมุดตั้งแต่ 1846 ส่วนการท้าในที่สาธารณะเกิดปี 1854 เล่มนี้อ่านแค่การอ้างอิงถึง Franksen ไม่ได้อ่านตัวเล่ม

หลักฐานทั้งหมดฝั่ง Babbage อยู่ในเอกสารที่ไม่เคยพิมพ์ ต้นฉบับเรื่อง cyphers and deciphering อยู่ที่ British Library Ole Immanuel Franksen อ่านมัน แล้วพิมพ์ Mr Babbage's Secret ปี 1985 กับบทความใน Mathematics and Computers in Simulation ปี 1993 ข้อค้นพบคือ Babbage มีของสามอย่างอยู่ในมือก่อนที่ใครจะพิมพ์ กฎคณิตศาสตร์ของรหัส หลักความเป็นคาบของกุญแจ และเทคนิคสมมาตรของตำแหน่ง เขาวางแผนหนังสือชื่อ The Philosophy of Decyphering ไว้ด้วย และไม่เคยเขียน

Babbage ถึงก่อน Kasiski เป็นเจ้าของผล ลำดับก่อนหลังในสมุดบันทึกไม่ใช่ลำดับก่อนหลังในวิชา เพราะการเจาะที่ไม่มีใครอ่านได้ไม่ป้องกันใครและไม่สอนใคร ทุกคนที่ใช้รหัสนี้ระหว่างปีที่ Babbage เจาะได้กับปี 1863 ส่งความลับของตัวเองผ่านระบบที่แตกไปแล้ว โดยที่ความมั่นใจของพวกเขาไม่ได้ลดลงแม้แต่วันเดียว นี่คือข้อสองของ Kerckhoffs มองจากอีกด้าน ระบบไม่ได้ปลอดภัยเพราะยังไม่มีใครพิมพ์วิธีเจาะ มันแค่ยังไม่ถูกพิมพ์

สิ่งที่กรณีนี้สอน: ตรวจว่าคำว่า unbreakable ที่ติดอยู่กับระบบของคุณมาจากการโจมตีที่ตีพิมพ์แล้วล้มเหลว หรือมาจากการที่ยังไม่มีใครตีพิมพ์ — ในกรณีนี้สองอย่างนั้นห่างกันอย่างน้อยเก้าปี

Enigma · 1932–1945

ทุกเช้า ผู้ส่งของเยอรมันจะตั้งโรเตอร์ของ Enigma ตามค่าประจำวัน แล้วทำสิ่งที่คู่มือสั่งไว้ในบรรทัดถัดมา คือเลือกกุญแจข้อความสามตัวอักษรของตัวเอง แล้วเข้ารหัสมันสองครั้งติดกัน KYG กลายเป็น KYGKYG ตัวเครื่องไม่ใช่สิ่งที่แตก เครื่องแข็ง สิ่งที่เปิดประตูคือบรรทัดนั้น

เพราะการซ้ำสองครั้ง Rejewski รู้ล่วงหน้าว่าในทุก indicator ตัวที่หนึ่งกับสี่มาจากตัวอักษรเดียวกัน สองกับห้าเหมือนกัน สามกับหกเหมือนกัน เก็บ indicator ทั้งวันมากองไว้ก็ได้โครงสร้างวงจรของ permutation ประจำวันนั้นออกมา จุดอ่อนอยู่ในคู่มือ ไม่ได้อยู่ในสายไฟ

คณิตศาสตร์ตามมาบนโครงสร้างนั้น Rejewski ใช้ทฤษฎีกลุ่มแทนภาษาศาสตร์ ทฤษฎีบทที่รับน้ำหนักคือ permutation สองตัว conjugate กันก็ต่อเมื่อมีโครงสร้างวงจรเหมือนกัน แล้วมันชนกำแพง สมการหกตัวมีตัวไม่รู้ค่ามากเกิน Rejewski พูดในปี 1980 ว่าจนวันนั้นเขาก็ยังไม่รู้ว่าสมการชุดนั้นแก้ได้หรือไม่ถ้าไม่มีข้อมูลเพิ่ม

ข้อมูลเพิ่มมาถึงราววันที่ 9 หรือ 10 ธันวาคม 1932 Marian Rejewski ได้รับเอกสารตั้งค่า Enigma ของเดือนกันยายนและตุลาคมปีนั้น สองเดือนที่คร่อมรอบเปลี่ยนลำดับโรเตอร์พอดี เอกสารมาจาก Hans-Thilo Schmidt สายลับในหน่วยรหัสเยอรมัน ผ่าน Gustave Bertrand ของฝรั่งเศส คณิตศาสตร์ที่สวยที่สุดในเรื่องนี้เดินต่อได้เพราะมีคนขโมยกระดาษมาให้

จากนั้นเป็นการแข่งเรื่องขั้นตอน ทุกวิธีของโปแลนด์ตายเพราะเยอรมันเปลี่ยนขั้นตอน ไม่ใช่เพราะใครหาคณิตศาสตร์ที่ดีกว่าได้ แคตตาล็อก 105,456 รายการที่ใช้เวลาสร้างกว่าหนึ่งปี ตายในคืนที่ reflector เปลี่ยนต้นเดือนพฤศจิกายน 1937 เดือนมกราคม 1938 โปแลนด์อ่านได้ 75% ของที่ดัก วันที่ 15 ธันวาคม 1938 โรเตอร์เพิ่มอีกสองตัว ถ้าจะไล่ให้ทันต้องใช้ bomba หกสิบเครื่อง ราคาสิบห้าเท่าของงบอุปกรณ์ทั้งปีของสำนัก การเจาะแพ้เลขคณิตก่อนแพ้การถอดรหัส

ที่ Bletchley เรื่องเดิมเกิดซ้ำ วันที่ 1 พฤษภาคม 1940 เยอรมันเลิกส่งกุญแจซ้ำสอง Hut 6 มืดภายในวันเดียว สิ่งที่ต่อชีวิตให้ไม่ใช่คณิตศาสตร์ แต่คือคนที่นั่งใช้เครื่องอยู่อีกฝั่ง John Herivel ไม่ได้มองหาข้อบกพร่องในวงจร เขามองหาสิ่งที่คนขี้เกียจจะทำ

ตอนต้นกะ และเห็นตั้งแต่กุมภาพันธ์ 1940 ว่าผู้ส่งแบบนั้นจะปล่อยโรเตอร์ไว้ใกล้ค่าตั้งวงแหวน แล้วใช้มันเป็นค่าตั้งต้นของข้อความแรกของวัน วางข้อความแรกของวันลงตาราง 17,576 ทางเลือกเหลือราว 6 ถึง 30 กลุ่มนั้นปรากฏขึ้นจริงวันที่ 10 พฤษภาคม และวันที่ 22 พฤษภาคมข้อความ Luftwaffe ถูกอ่านได้อีกครั้ง วิธีนี้กับ cillies ถือแนวไว้จน bombe มาถึงเดือนสิงหาคม

ข้อบกพร่องในตัวเครื่องมีข้อเดียว ตัวอักษรไม่เคยเข้ารหัสเป็นตัวเอง มันไม่ได้เจาะอะไรด้วยตัวมันเอง แต่ทำให้ crib ถูก สัดส่วนระหว่างคณิตศาสตร์กับขั้นตอนไม่มีตัวเลข สิ่งที่หลักฐานรับได้คือคณิตศาสตร์จำเป็นและไม่เคยพอ ความพอมาจากคนที่ใช้ระบบ ทุกครั้ง

สิ่งที่กรณีนี้สอน: ให้คนที่จะเจาะระบบของคุณอ่านคู่มือปฏิบัติงานก่อนอ่านแบบวงจร — จุดเข้าของ Enigma ทุกจุดที่มีบันทึกอยู่ในขั้นตอนหรือในคน ไม่ใช่ในสายไฟ

โทรเลข Zimmermann · 1917

อังกฤษถือกระดาษที่ดึงอเมริกาเข้าสงครามได้อยู่ในมือตั้งแต่วันรุ่งขึ้นหลังดักได้ แล้วรอต่ออีกหกสัปดาห์ ไม่ใช่เพราะถอดไม่ออก แต่เพราะการหยิบมันออกมาใช้จะบอกอีกฝ่ายไปพร้อมกันว่าได้มันมาอย่างไร ข่าวกรองที่ใช้แล้วเผาแหล่งของตัวเอง มีค่าเท่ากับข่าวกรองที่ใช้ได้ครั้งเดียวในชีวิต

วันที่ 17 มกราคม 1917 กระทรวงต่างประเทศเยอรมันส่งโทรเลขในนาม Zimmermann ถึงทูต Heinrich von Eckardt ที่เม็กซิโกในรหัสการทูต 13040 Room 40 ของอังกฤษดักได้ และวันรุ่งขึ้น Nigel de Grey ถอดได้บางส่วน จากวันนั้นถึงวันพิมพ์ใช้เวลาราวหกสัปดาห์ ถึงวันที่สภาคองเกรสประกาศสงครามราวสิบเอ็ดสัปดาห์

ปัญหาอยู่ที่สายที่โทรเลขวิ่งมา กระทรวงต่างประเทศสหรัฐยอมให้เยอรมันใช้สายของตัวเองส่งข้อความเข้ารหัสเพื่อการเจรจาสันติภาพ และฉบับนี้ยังถูกส่งทางวิทยุกับซ่อนมาในโทรเลขทางการทูตของสวีเดนบนสายเคเบิลของอังกฤษด้วย พิมพ์ทันทีจึงเท่ากับบอกเยอรมันว่ารหัสแตก และบอกอเมริกาว่าอังกฤษอ่านโทรเลขทางการทูตของอเมริกาเองอยู่ Hall รอราวสามสัปดาห์ระหว่างที่ de Grey กับ William Montgomery ถอดจนครบ

ทางออกไม่ใช่การซ่อนแหล่งให้มิดขึ้น แต่คือการหาแหล่งที่สองที่ยอมเผาได้ สถานทูตเยอรมันที่วอชิงตันส่งต่อโทรเลขไปเม็กซิโกทางบริษัทโทรเลขพาณิชย์ในรหัส 13040 เดิม สายลับอังกฤษในเม็กซิโกที่บันทึกไว้แค่ว่า Mr. H ติดสินบนพนักงานบริษัทโทรเลขจนได้ตัวรหัสฉบับนั้นมา อังกฤษจึงมีข้อความครบจากแหล่งที่บอกชื่อได้ กรณีเลวร้ายที่สุดที่เหลืออยู่คือเยอรมันสรุปว่า 13040 ถูกอ่าน และราคานั้นคุ้มกับการดึงอเมริกาเข้าสงคราม ฉบับที่เล่ากันว่าโทรเลขต้นทางส่งในรหัสรุ่นใหม่กว่าและมีเลขรหัสอีกชุดเล่มนี้ยืนยันเลขนั้นกับแหล่งที่อ่านไม่ได้ จึงไม่พิมพ์

จากนั้นอังกฤษทำสิ่งที่ยากกว่าการโน้มน้าว คือให้อเมริกาตรวจเองโดยไม่ต้องเชื่ออังกฤษ 19 กุมภาพันธ์ Hall ให้ Edward Bell เลขานุการสถานทูตอเมริกันในลอนดอนดู Bell คิดว่าปลอมก่อน แล้วโกรธ 23 กุมภาพันธ์ Balfour มอบตัวรหัส ข้อความเยอรมัน และคำแปลอังกฤษให้ทูต Page พร้อมความจริงครึ่งเดียวว่าซื้อมาจากเม็กซิโก 24 กุมภาพันธ์ Page รายงาน Wilson พร้อมรายละเอียดที่อเมริกาตรวจได้กับแฟ้มของบริษัทโทรเลขของตัวเอง ส่วนตัวอังกฤษต้องมอบรหัส 13040 ให้วอชิงตันถอดเองด้วย

พิมพ์วันที่ 1 มีนาคม วันที่ 3 มีนาคม Zimmermann บอกนักข่าวอเมริกันว่าปฏิเสธไม่ได้ มันเป็นความจริง ปัญหาความแท้ที่อังกฤษเตรียมรับมือมาหกสัปดาห์ ตายด้วยมือผู้ส่งเองในประโยคเดียว 29 มีนาคมเขาบอก Reichstag ว่าเอกสารหลุดไปในทางที่ไม่ปราศจากข้อครหา เยอรมันสรุปว่าถูกขโมย ไม่ใช่ถูกถอด Lansing ช่วยด้วยการปล่อยคำใบ้เรื่องสายลับ 6 เมษายนคองเกรสประกาศสงคราม

สิ่งที่กรณีนี้สอน: ก่อนใช้สิ่งที่ได้จากช่องทางลับ ให้เอาของชิ้นเดียวกันผ่านช่องทางที่สองที่เปิดเผยได้ แล้วส่งคำอธิบายที่อีกฝ่ายตรวจเองได้ไปก่อนที่เขาจะถามว่าคุณรู้ได้อย่างไร

VENONA · 1942–1980

one-time pad คือรหัสตัวเดียวที่มีบทพิสูจน์ว่าเจาะไม่ได้ และบทพิสูจน์นั้นไม่เคยผิด สิ่งที่มันพิงอยู่เงียบๆ คือข้อสมมติข้อเดียว ว่าทุกหน้าของแผ่นกุญแจถูกใช้ครั้งเดียวแล้วทิ้ง ข้อสมมตินั้นไม่ได้อยู่ในคณิตศาสตร์ มันอยู่ในโรงพิมพ์

วันที่ 1 กุมภาพันธ์ 1943 Signal Intelligence Service ของกองทัพบกสหรัฐเปิดโครงการอ่านโทรเลขโซเวียตที่ Arlington Hall และปิดวันที่ 1 ตุลาคม 1980 สามสิบเจ็ดปี ได้ข้อความออกมาราว 3,000 ฉบับจากที่ดักได้หลายแสน

ระบบที่มันเจาะใช้สมุดรหัสแปลงคำเป็นตัวเลข แล้วบวกทับด้วย one-time pad ใช้ถูกวิธี ไม่มีคณิตศาสตร์ใดถอดได้ สิ่งที่แตกจึงไม่ใช่วิธี แต่คือโรงงาน ผู้ผลิตแผ่นกุญแจของโซเวียตพิมพ์หน้าซ้ำออกมาราว 35,000 หน้า ทั้งหมดในปี 1942 ระหว่างที่กองทัพเยอรมันรุกเข้าหามอสโก การผลิต pad ช้าและทำด้วยมือ ความต้องการพุ่งหลังมิถุนายน 1941 คนที่นั่งพิมพ์อยู่ปีนั้นไม่ได้กำลังตัดสินใจเรื่องความมั่นคง เขากำลังพยายามส่งของให้ทัน โซเวียตรู้ และพยายามคุมด้วยการส่งหน้าซ้ำไปให้ผู้ใช้ที่อยู่ห่างกันมาก ไม่สำเร็จ ที่ซ้ำคือหน้า ไม่ใช่ทั้งเล่ม

ร้อยโท Richard Hallock ที่ทำงานกับโทรเลขการค้าของโซเวียตเห็นการซ้ำเป็นคนแรก เขากับ Genevieve Feinstein, Cecil Phillips, Frank Lewis, Frank Wanat และ Lucille Campbell กู้ตารางกุญแจได้จำนวนมาก Meredith Gardner สร้างสมุดรหัสขึ้นใหม่ และเจาะเข้าตัวรหัสได้ครั้งแรกวันที่ 20 ธันวาคม 1946 สิ่งที่เขาเห็นในข้อความชุดนั้นคือสายลับโซเวียตใน Manhattan Project

ตัวเลขต้องอ่านให้ตรง อัตราที่อ่านได้ของโทรเลข NKVD คือ 1.8% ของปี 1942, 15.0% ของปี 1943, 49.0% ของปี 1944 และ 1.5% ของปี 1945 โทรเลข GRU ทหารเรือสายวอชิงตัน–มอสโกปี 1943 อ่านได้ราวครึ่ง ปีอื่นอ่านไม่ได้เลย ถอดและแปลได้ราว 2,200 ฉบับ ฉบับที่ว่า VENONA เจาะการสื่อสารของโซเวียตได้ จึงหมายถึงราวหนึ่งเปอร์เซ็นต์ของโทรเลข กระจุกในสองปี และมีอยู่เพราะข้อบกพร่องในการผลิตของปีเดียว

หน้าซ้ำเกือบทั้งหมดถูกใช้หมดภายในปลายปี 1945 บางหน้าใช้ถึงปี 1948 หลังจากนั้นโทรเลขกลับไปอ่านไม่ได้เลย Bill Weisband สายลับ NKVD ในหน่วยดักฟังของกองทัพบกสหรัฐบอกมอสโกเรื่อง VENONA ในปี 1945 Kim Philby ได้รู้ปี 1949 โซเวียตไม่เปลี่ยนรหัสอะไรเลย เพราะไม่มีอะไรต้องเปลี่ยน หน้าซ้ำถูกใช้ไปหมดแล้ว พวกเขาแค่เตือนสายลับที่เสี่ยง ผลจากหนึ่งเปอร์เซ็นต์นั้นคือ Cambridge Five, Klaus Fuchs, Donald Maclean, สายลับใน State, Treasury, OSS และทำเนียบขาว และคดี Rosenberg ส่วน Roosevelt กับ Truman ไม่เคยได้รับรายงาน

สิ่งที่กรณีนี้สอน: ถ้าระบบของคุณพิสูจน์ได้ว่าเจาะไม่ได้ ให้ไปดูว่าใครผลิตกุญแจผลิตเร็วแค่ไหน และเกิดอะไรขึ้นในเดือนที่ความต้องการเกินกำลังผลิต — key reuse ไม่เคยอยู่ในบทพิสูจน์

The Thing · 1945–1952

วิธีหาเครื่องดักฟังทั้งหมดในปี 1945 สร้างขึ้นบนข้อเท็จจริงข้อเดียว คือเครื่องดักฟังต้องใช้ไฟ ไฟแปลว่ามีแบตเตอรี่หรือมีสายที่เดินไปหาได้ มีความร้อน และมีคลื่นแผ่ออกมาให้เครื่องกวาดจับ ของที่แขวนอยู่ในห้องทำงานทูตอเมริกันที่มอสโกเจ็ดปีไม่มีสักข้อเดียวในนั้น

วันที่ 4 สิงหาคม 1945 คณะเยาวชน Young Pioneers ของโซเวียตมอบตราแผ่นดินสหรัฐแกะจากไม้ให้ทูต W. Averell Harriman เป็นของขวัญแก่พันธมิตรร่วมสงคราม มันแขวนในห้องทำงานทูตที่ Spaso House กรุงมอสโกนับจากวันนั้นเจ็ดปี พร้อมเครื่องดักฟังหนัก 31 กรัมที่ไม่มีแบตเตอรี่อยู่ข้างใน

เครื่องนี้ออกแบบโดย Leon Theremin ชื่อรัสเซียว่า Zlatoust และเป็น passive resonant cavity เสาอากาศ monopole ยาว 23 เซนติเมตรทะลุเข้าโพรงทองแดงชุบเงิน ปิดหน้าด้วยเยื่อนำไฟฟ้าหนา 75 ไมโครเมตร เยื่อกับแท่งปรับจูนด้านในรวมกันเป็นตัวเก็บประจุแปรค่า นั่นคือไมโครโฟนคอนเดนเซอร์ เสียงพูดสั่นเยื่อ ค่าความจุเปลี่ยนตามการสั่นนั้น คลื่นวิทยุที่ตกกระทบถูก modulate แล้วสะท้อนกลับเป็นฮาร์มอนิกสูงกว่าความถี่ที่ส่องมา พลังงานทุกหน่วยในเรื่องนี้มาจากฝั่งคนดักฟัง ไม่มีสักหน่วยที่มาจากตัวเครื่อง ไม่มีแบตเตอรี่ ไม่มีชิ้นส่วน active ไม่มีอะไรสึก ความถี่ที่โซเวียตใช้ส่องจริงยืนยันไม่ได้ แหล่งให้ 330 MHz เป็นค่าที่น่าจะที่สุด

ผลตามมาคือมันเงียบจนกว่าจะมีใครส่องคลื่นใส่ NSA ใช้คำว่า illuminating กับคลื่นที่ส่องเข้าไปนั้น กวาดหาการแผ่คลื่นแล้วเจอศูนย์ เพราะไม่มี oscillator ไม่กินไฟ ไม่มีความร้อน ไม่มีสาย ตัวมันเล็ก และมันอยู่ในของขวัญที่รับกันต่อหน้าคน จะพบมันได้จึงต้องบังเอิญ หรือต้องรุก

ปี 1951 พนักงานวิทยุอังกฤษที่สถานทูตอังกฤษในมอสโกได้ยินบทสนทนาของอเมริกันบนช่องวิทยุกองทัพอากาศโซเวียตโดยบังเอิญ ระหว่างที่โซเวียตกำลังส่อง

คลื่นใส่ห้องทูตอยู่พอดี เดือนมีนาคมปีนั้น John W. Ford กับ Joseph Bezjian ถูกส่งไปมอสโก วิธีของพวกเขาไม่ใช่ฟัง แต่คือส่ง เครื่องกำเนิดสัญญาณกับเครื่องรับถูกจัดให้ห้องหอนด้วย feedback ถ้าเสียงในห้องถูกส่งซ้ำออกมาบนความถี่นั้น Bezjian พบมันในตราไม้ เครื่องกำเนิดที่พบมันตั้งไว้ที่ 1800 MHz เรื่องถึงผู้นำสถานทูตในสมัยทูต George F. Kennan ปี 1952

Peter Wright จาก Marconi ที่ต่อมาอยู่ MI5 เป็นคนวิเคราะห์ เยื่อขาดระหว่างอเมริกันจับต้อง เขาต้องเปลี่ยนใหม่ และทำให้มันทำงานเสถียรที่ 800 MHz ตัวแปรที่ตัดสินคือ Q ของโพรง Q สูงหมายถึงเลือกความถี่ได้แหลมกว่า ระยะไกลกว่า และไวต่อเสียงกว่า ผลคือ SATYR รุ่นอังกฤษที่ใช้ตลอดทศวรรษ 1950 เดือนพฤษภาคม 1960 ทูต Henry Cabot Lodge Jr. ยกตราไม้ขึ้นกลางคณะมนตรีความมั่นคงสหประชาชาติระหว่างเรื่อง U-2

สิ่งที่กรณีนี้สอน: เขียนว่าเครื่องตรวจของคุณตรวจหาคุณสมบัติอะไร แล้วออกแบบสิ่งที่ไม่มีคุณสมบัตินั้นเลย — ถ้าคุณออกแบบได้ ศัตรูก็ทำได้ และการตรวจต้องเปลี่ยนจากฟังเป็นส่อง

Van Eck · 1985

คนที่นั่งพิมพ์อยู่หน้าจอในปี 1985 เชื่อกันโดยไม่ต้องคิดว่าถ้าไม่มีใครยืนอยู่ข้างหลัง ก็ไม่มีใครอ่านสิ่งที่ขึ้นบนจอนั้นได้ ความเชื่อนั้นผิดมาแล้วสี่สิบปี และคนที่รู้ว่ามันผิดพูดไม่ได้ สิ่งที่ Wim van Eck เปลี่ยนในปีนั้นจึงไม่ใช่ความรู้ แต่คือใครมีมัน

ปี 1985 Wim van Eck จาก Dr. Neher Laboratories ของ PTT เนเธอร์แลนด์พิมพ์ใน Computers & Security เล่มที่ 4 ว่าจอ CRT ในกล่องพลาสติกอ่านได้ชัดบนโทรทัศน์ธรรมดาที่ระยะราว 50 เมตรในสนามทดสอบ CISPR และอุปกรณ์เสริมที่ต้องต่อเข้าโทรทัศน์ราคาราว 15 ดอลลาร์ นี่คือการวิเคราะห์ที่ไม่ลับชิ้นแรกของความเสี่ยงนี้

กลไกอยู่ในแบบของจอเอง สัญญาณวิดีโอเป็นสัญญาณเดียวในจอที่ถูกขยายจากระดับ TTL ไปถึงหลายร้อยโวลต์ มันจึงครองสนามคลื่นที่แผ่ออกมา และฮาร์มอนิกของมันคล้ายสัญญาณโทรทัศน์ออกอากาศมากพอที่เครื่องรับธรรมดาจะจับได้ สิ่งที่

ขาดคือ sync ภาพจึงเลื่อนไม่หยุด คืน sync ด้วย oscillator สองตัวแล้วภาพก็นิ่ง นักอิเล็กทรอนิกส์สมัครเล่นสร้างของชิ้นนั้นได้ในไม่กี่วัน

ตัวเลขที่วัดกับตัวเลขที่ประมาณต้องแยกกัน ที่วัดได้คือราว 50 เมตรสำหรับกล่องพลาสติก ราว 10 เมตรสำหรับกล่องโลหะ ที่ประมาณคือถึง 1 กิโลเมตรถ้าใช้เสาอากาศทิศทางที่ได้เกนอย่างน้อย 10 dB ข้อสรุปข้อสองในบทความเขียนตรงตัวว่าอาจทำได้ under optimum conditions จากระยะถึง 1 กิโลเมตร ฉบับที่เล่ากันว่า van Eck อ่านจอจากหนึ่งกิโลเมตรด้วยของ 15 ดอลลาร์ จึงผิดสามชั้นซ้อนกัน และบรรทัดที่ควรอ่านช้าๆ คือบรรทัดนี้ จอทุกเครื่องที่ทดสอบผ่านเกณฑ์รบกวน CISPR ที่เสนอในตอนนั้น และทุกเครื่องรั่วภาพ

การสาธิตมีสองครั้ง ครั้งแรกจากรถในลานจอดของอาคารที่มีเครื่องประมวลผลคำอยู่ข้างใน ครั้งที่สองเดือนกุมภาพันธ์ 1985 ที่ลอนดอนกับ BBC รถตู้ เสาสูง 10 เมตร เสาอากาศ VHF band III เกน 10 dB เครื่องขยาย 18 dB โทรทัศน์ในรถ ทำกลางวันต่อหน้าคนดู และไม่มีใครถามว่ากำลังทำอะไร

เขาเอ่ยถึงมาตรฐานสองฉบับที่เขาเองอ่านไม่ได้ NACSIM 5100A และ AMSG 720B รัฐบาลรู้มาสี่สิบปี Bell Labs สาธิตต่อ Signal Corps ในสงครามโลกครั้งที่สองที่ถนน Varick ในแมนแฮตตัน DoD Directive 5200.19 ลงนามธันวาคม 1964 ตลอดช่วงนั้นผู้ใช้จอทั่วไปไม่มีส่วนเผื่อเลย เพราะแม้แต่ตัวภัยก็เป็นความลับ นี่คือข้อสองของ Kerckhoffs มาถึงช้า 102 ปี จากฝั่งผู้ป้องกัน ทางแก้ที่เขาเสนอคือ cryptographic display เขียนบรรทัดบนจอในลำดับที่ขึ้นกับกุญแจ ราคาราว 20 ดอลลาร์ต่อเครื่อง

สิ่งที่กรณีนี้สอน: ถ้ามาตรฐานที่ป้องกันคุณเป็นความลับ ให้ถือว่าคุณไม่มีมาตรฐาน — วัดระยะที่จอของคุณอ่านได้ด้วยโทรทัศน์กับเสาอากาศ ก่อนเชื่อใบรับรองว่าผ่านค่ารบกวน

ท่าที่เอาไปใช้ได้: สมมติว่าอีกฝ่ายถือแบบแปลนฉบับเต็มของระบบคุณ รวมคู่มือปฏิบัติงานและวิธีที่คุณใช้ตรวจหาเขา แล้วเขียนว่าอะไรยังป้องกันได้ ถ้าคำตอบยาวกว่ากุญแจหนึ่งดอก ทุกบรรทัดที่เกินคือส่วนเผื่อที่มีอายุแค่จนกว่าจะมีคนอ่าน

จะพลิกคำตัดสินของภาคนี้ได้เมื่อ: ผู้อ่านชี้ได้จากแหล่งข้างล่างว่ากรณีใดในแปดกรณีนี้ถูกเจาะโดยผู้เจาะที่ไม่รู้แบบ ไม่รู้ขั้นตอน และไม่มีเอกสารที่ยึดหรือซื้อมาได้ —

เจาะด้วยกำลังคำนวณล้วนต่อระบบที่ยังปิดอยู่ กรณีนั้นออกจากภาคนี้ทันที และถ้าเกินสองกรณี บานพับของเล่มนี้ตั้งผิดที่

แหล่งของภาคนี้:

- Kerckhoffs — ข้อความเต็มภาษาฝรั่งเศส ตอนที่หนึ่ง · https://www.petitcolas.net/kerckhoffs/la_cryptographie_militaire_i.htm
- Kerckhoffs — รายละเอียดบรรณานุกรมและคำอธิบายของ Petitcolas · https://www.petitcolas.net/kerckhoffs/
- Kerckhoffs — สแกนต้นฉบับ ตอนที่หนึ่ง · https://www.petitcolas.net/kerckhoffs/crypto_militaire_1_b.pdf
- ลอนดอน 1851 — Rudimentary Treatise on the Construction of Locks ฉบับเต็มปี 1853 · https://gutenberg.org/files/63128/63128-h/63128-h.htm
- ลอนดอน 1851 — บทที่หนึ่งบน Wikisource · https://en.wikisource.org/wiki/Rudimentary_Treatise_on_the_Construction_of_Locks/Chapter_1
- ลอนดอน 1851 — Linda Hall Library, Alfred Charles Hobbs · https://www.lindahall.org/about/news/scientist-of-the-day/alfred-charles-hobbs/
- ลอนดอน 1851 — Bramah lock · https://en.wikipedia.org/wiki/Bramah_lock
- ลอนดอน 1851 — Chubb detector lock · https://en.wikipedia.org/wiki/Chubb_detector_lock
- le chiffre indéchiffrable — Kasiski examination · https://en.wikipedia.org/wiki/Kasiski_examination
- le chiffre indéchiffrable — Franksen 1993, Mathematics and Computers in Simulation · https://www.sciencedirect.com/science/article/abs/pii/037847549390063Z
- le chiffre indéchiffrable — MacTutor, Babbage and deciphering · https://mathshistory.st-andrews.ac.uk/Extras/Babbage_deciphering/
- Enigma — Marian Rejewski · https://en.wikipedia.org/wiki/Marian_Rejewski
- Enigma — John Herivel · https://en.wikipedia.org/wiki/John_Herivel
- Enigma — Hans-Thilo Schmidt · https://en.wikipedia.org/wiki/Hans-Thilo_Schmidt
- Enigma — NSA, Solving the Enigma: History of the Cryptanalytic Bombe · https://media.defense.gov/2022/Sep/29/2003087366/-1/-1/0/SOLVING%20THE%20ENIGMA%20-%20HISTORY%20OF%20THE%20CRYPTANALYTIC%20BOMBE.PDF
- Enigma — Cryptologia, Human factors and missed solutions to Enigma design weaknesses · https://www.tandfonline.com/doi/full/10.1080/01611194.2015.1028680
- Zimmermann — Zimmermann telegram · https://en.wikipedia.org/wiki/Zimmermann_telegram

- Zimmermann — GCHQ, บรรพบุรุษของ GCHQ กับการเข้าสงครามของสหรัฐ · https://www.gchq.gov.uk/information/century-how-work-gchqs-predecessors-contributed-us-entering-world-war-i
- Zimmermann — FRUS, Page ถึง Secretary of State, 3 มีนาคม 1917 · https://history.state.gov/historicaldocuments/frus1917Supp01v01/d181
- VENONA — Venona project · https://en.wikipedia.org/wiki/Venona_project
- VENONA — Meredith Gardner · https://en.wikipedia.org/wiki/Meredith_Gardner
- VENONA — Richard Hallock · https://en.wikipedia.org/wiki/Richard_Hallock_(Venona_Project)
- The Thing — Crypto Museum · https://www.cryptomuseum.com/covert/bugs/thing/index.htm
- The Thing — Murray Associates, Great Seal bug part 1 · https://counterespionage.com/great-seal-bug-part-1/
- The Thing — บันทึกคณะมนตรีความมั่นคง S/PV.860, 26 พฤษภาคม 1960 · https://undocs.org/en/S/PV.860
- The Thing — History of the Bureau of Diplomatic Security หน้า 136–137 · https://2009-2017.state.gov/documents/organization/176589.pdf
- Van Eck — บทความต้นฉบับปี 1985 · http://www.tscm.com/vaneck85.pdf
- Van Eck — DOI ของบทความ · https://doi.org/10.1016/0167-4048(85)90046-X
- Van Eck — Tempest (codename) · https://en.wikipedia.org/wiki/Tempest_(codename)
- Van Eck — Kuhn, Compromising emanations, 2003 · http://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-577.pdf

4 · เมื่อโครงสร้างเป็นซอฟต์แวร์

ภาคนี้เปลี่ยนวัสดุ และสิ่งที่เปลี่ยนไปพร้อมกับวัสดุคือการทำสำเนา สะพานที่ออกแบบผิดหนึ่งแห่งถล่มหนึ่งแห่ง ส่วนโค้ดที่เขียนผิดหนึ่งบรรทัดเดินทางไปถึงทุกเครื่องที่ติดตั้งมัน เพราะข้อบกพร่องในซอฟต์แวร์คัดลอกได้โดยไม่มีต้นทุน จุดอ่อนจุดเดียวจึงถูกส่งออกไปพร้อมกันเป็นพันล้านสำเนา และคนที่หาเจอคนแรกได้ทั้งพันล้านนั้นไปในคราวเดียว

ตัวเลขที่ภาคนี้ตามมีตัวเดียว — ข้อบกพร่องมีอยู่นานเท่าไรก่อนมีคนใช้มัน ในสิบสี่กรณีข้างล่าง คำตอบวัดเป็นปีเกือบทุกครั้ง ไม่ใช่วันหรือสัปดาห์ และช่องว่างนั้นเองคือเรื่องของภาคนี้

ก่อนเริ่ม มีข้อตกลงเรื่องตัวเลขหนึ่งข้อ ตัวเลขเงินในวงการนี้เป็นตัวเลขที่เชื่อถือได้น้อยที่สุดบนหน้ากระดาษ จำนวนบันทึกที่รั่วใกล้เคียงการตรวจสอบ ระยะเวลาที่ผู้โจมตีอยู่ในระบบมาจากไทม์ไลน์นิติวิทยาศาสตร์และใช้ได้ แต่ตัวเลขดอลลาร์เกือบทั้งหมดคือแถลงข่าว คำฟ้อง หรือการเดาของนักวิเคราะห์ที่ถูกเล่าซ้ำจนแข็ง เล่มนี้พิมพ์ทั้งตัวเลขที่บริษัทเปิดเผยเองและตัวเลขพาดหัว และบอกว่าอันไหนเป็นอันไหน ตัวเลขดังหลายตัวที่เล่ากันติดปากจะไม่อยู่ในหน้าถัดไป และทุกครั้งจะมีหนึ่งบรรทัดบอกว่าทำไมมันหายไป

Multics · 1974

ในปี 1974 ถ้าถามว่าระบบปฏิบัติการที่มั่นคงที่สุดในตลาดคือตัวไหน คำตอบที่วงการให้ตรงกันมีชื่อเดียวคือ Multics และคำถามที่ศูนย์ข้อมูลของกองทัพอากาศในชั้นใต้ดินเพนตากอนอยากได้คำตอบก็ตรงไปตรงมา — Multics บนเครื่อง Honeywell HIS 645 และ HIS 6180 รับงาน Secret กับ Top Secret บนเครื่องเดียวกันได้หรือไม่

เดือนมิถุนายน 1974 กองทัพอากาศสหรัฐตีพิมพ์คำตอบในรายงาน ESD-TR-74-193 เล่มที่สอง ผู้เขียนคือ Paul Karger และ Roger Schell และคำตัดสินคือรับรองให้ใช้กับงานสองระดับชั้นความลับไม่ได้

ทีมพบข้อบกพร่องสามชั้น ฮาร์ดแวร์ ซอฟต์แวร์ และขั้นตอนปฏิบัติ ในชั้นซอฟต์แวร์มีโมดูลที่ออกแบบให้ทำงานในระดับผู้ใช้ แต่กลับทำงานด้วยสิทธิ์ระดับระบบ นี่คือรายงานเจาะระบบปฏิบัติการเชิงพาณิชย์ฉบับแรกที่ตีพิมพ์โดยละเอียด ไม่ใช่การเจาะระบบครั้งแรก ทีมแบบนี้มีมาตั้งแต่ปลายทศวรรษ 1960 สิ่งที่ใหม่คือมีคนพิมพ์วิธีการออกมาให้คนนอกอ่าน

แต่รายการข้อบกพร่องไม่ใช่ส่วนที่สำคัญที่สุด และผู้เขียนทั้งสองเขียนไว้เองในบทย้อนหลังปี 2002 ว่ามันเป็นครึ่งเล็กของผลลัพธ์ ครึ่งใหญ่คือการสาธิตว่ากลไกป้องกันถูกล้มได้ด้วยซอฟต์แวร์ที่ตั้งใจหลอกมัน — trap door และ Trojan horse ความต่างนี้สำคัญเพราะมันเปลี่ยนว่าอะไรคือทางแก้ แพตช์ปิดข้อบกพร่องที่หาเจอแล้ว มันไม่ได้ปิดช่องทางที่ยังไม่มีใครไปหา ข้อสรุปของรายงานจึงไม่ใช่รายการแพตช์ แต่คือคำว่าต้องรื้อโครงสร้างใหม่รอบแกนความมั่นคงที่พิสูจน์ได้ ก่อนใช้ระบบใดในสภาพแวดล้อมเปิดที่มีมืออาชีพเป็นศัตรู

วิธีที่ทีมใช้พิสูจน์ข้อนั้นคือสิ่งที่ทำให้กรณีนี้ยังมีค่าอยู่จนวันนี้ พวกเขาไม่ได้แค่หาข้อบกพร่อง พวกเขาปลูกมันด้วย trap door ตัวหนึ่งถูกฝังในเครื่อง 645 และไม่เคยถูกแจกจ่าย เพราะเครื่องรุ่นนั้นกำลังถูกปลด ตัวที่สองถูกฝังในระบบพัฒนาของเครื่อง 6180 ที่ MIT จากจุดนั้นมันไม่ต้องทำอะไรเลย มันแค่เดินทางไปตามเส้นทางปกติที่ซอฟต์แวร์ทุกตัวเดิน ฝ่ายควบคุมคุณภาพของ Honeywell ตรวจไม่พบ และ Honeywell ส่งมันไปกับซอฟต์แวร์ถึงเครื่องของกองทัพอากาศเอง

ตัวเลขของกรณีนี้อยู่ตรงนี้ trap door ตัวที่สองอยู่รอดราวหนึ่งปีหลังรายงานตีพิมพ์ ทั้งที่รายงานพิมพ์คำใบ้ตำแหน่งของมันไว้ในมาตรา 3.4.6 มันถูกพบระหว่างการตรวจสอบเข้มที่ศูนย์ Multics ของ General Motors เมื่อรุ่นที่ถูกถอดเขี้ยวแล้วคืนค่าข้อผิดพลาดแปลกออกมา ห่างจากการทำงานจริงหนึ่งคำสั่ง อ่านลำดับนั้นอีกรอบแล้วจะเห็นว่าอะไรล้มเหลว คำเตือนถูกพิมพ์ไว้แล้ว พร้อมที่อยู่ของสิ่งที่ต้องไปดู และยังไม่พอให้ใครไปหา

มาตรา 3.2.4 ของรายงานฉบับเดียวกันทำนาย trap door ไว้สามแบบ ในสายแจกจ่ายซอฟต์แวร์ ในขั้นติดตั้ง และใน compiler ที่ฝัง trap door ลงในทุกอย่างที่มันแปล ปี 1974 ทั้งสามข้อยังเป็นคำทำนาย ไม่ใช่รายงานเหตุการณ์

สิ่งที่กรณีนี้สอน: ทดสอบกลไกป้องกันด้วยการปลูกของปลอมลงไปเองแล้วนับวัน จนกว่าฝ่ายควบคุมคุณภาพจะเจอ — ตัวเลขที่ได้คือส่วนเผื่อจริง ไม่ใช่ใบรับรอง

Trusting Trust · 1984

ข้อที่ Ken Thompson ขึ้นไปพังบนเวทีรับรางวัล Turing คือข้อที่ฟังดูไม่มีอะไรให้เถียง ถ้าอยากรู้ว่าโปรแกรมตัวหนึ่งทำอะไร ก็เปิด source ออกมาอ่าน เดือนสิงหาคม 1984 Communications of the ACM ฉบับที่ 27(8) พิมพ์ปาฐกถานั้นยาวสามหน้า หน้า 761 ถึง 763 และจำนวนเครื่องที่เสียหายจากมันคือศูนย์

กรณีนี้อยู่ในภาคเพราะมันไม่มีผู้เสียหาย ข้ออ้างของ Thompson ไม่ใช่ว่า compiler มีบั๊กได้ ทุกคนรู้อยู่แล้วว่ามีได้ มันคือโครงสร้างสามขั้นที่ทำให้การอ่านไร้ความหมาย ขั้นแรก สอน binary ของ compiler ให้จำโปรแกรม login ได้ และใส่ทางลับลงไปเมื่อแปลมัน ขั้นสอง สอนมันให้จำตัวเองได้ และใส่พฤติกรรมทั้งสองกลับเข้าไปใน compiler ทุกตัวที่มันแปล ขั้นสาม ลบ source ส่วนที่เป็นพิษทิ้ง

จากจุดนั้นทุกอย่างที่ตรวจได้ก็สะอาดหมด source ของ compiler สะอาด source ของ login สะอาด compiler ทุกตัวที่สร้างจาก source สะอาดถือข้อบกพร่องต่อไป และการอ่าน source กี่รอบก็ไม่เจอ เพราะสิ่งที่ต้องอ่านไม่ได้อยู่ในสิ่งที่อ่านได้ ประโยคสรุปของเขา: คุณไว้ใจโค้ดที่คุณไม่ได้สร้างเองทั้งหมดไม่ได้ อ่าน source ไม่เท่ากับไว้ใจโปรแกรม เพราะโปรแกรมมาจากเครื่องมือ และเครื่องมือสร้างตัวเองซ้ำ

ที่มาของความคิดมักถูกเล่าผิด และคนที่เล่าถูกที่สุดคือ Thompson เอง เขาเขียนว่าได้แนวคิดจากเอกสารกองทัพอากาศที่ไม่ทราบชื่อ และขอให้ใครก็ได้ช่วยหาที่อ้างอิงที่ดีกว่า Karger และ Schell บันทึกว่าเอกสารนั้นคือรายงาน Multics ของพวกเขาเองปี 1974 พวกเขาส่งสำเนาให้เขาหลังปาฐกถาตีพิมพ์ และ Thompson แก้ที่อ้างอิงในฉบับพิมพ์ซ้ำใน Unix Review เดือนพฤศจิกายน 1989 ความคิดเป็นของปี 1974 การสาธิตและชื่อเสียงเป็นของปี 1984 ส่วนที่ว่า compiler ตัวนั้นเคยถูกใช้นอก Bell Labs หรือไม่ มีคนยืนยันทั้งสองทาง และเล่มนี้ยืนยันไม่ได้

คำถามประจำภาค — ข้อบกพร่องมีอยู่นานเท่าไรก่อนถูกใช้ — ตอบแบบเดิมไม่ได้ในกรณีนี้ เพราะสิ่งที่ Thompson ชี้ไม่ใช่ข้อบกพร่องที่ถูกเพิ่มเข้าไปวันหนึ่ง เงื่อนไขที่

เขาอธิบายคือการไว้ใจ binary ของเครื่องมือ และมันมีมาตั้งแต่ compiler เริ่มแปลตัวเอง การอ่านตรวจแก้มันไม่ได้ วิธีรับมือมาถึงยี่สิบเอ็ดปีให้หลัง David A. Wheeler เสนอ Diverse Double-Compiling ตรวจ binary ของ compiler เทียบกับ binary ที่สร้างจาก source เดียวกันด้วย compiler อิสระอีกตัว ปี 2023 Russ Cox สร้างการโจมตีต้นฉบับของ Thompson ขึ้นใหม่และรันมันได้ สี่สิบปีจากคำเตือนถึงการยืนยันว่ามันทำงานจริง

ในแง่สันของเล่ม นี่คือกรณีที่ศัตรูไม่ได้อ่านแบบก่อสร้าง เขาอ่านเครื่องมือที่ใช้อ่านแบบ

สิ่งที่กรณีนี้สอน: เมื่อตรวจ source ให้ถามต่อว่าใครตรวจ compiler ที่แปลมัน และตรวจด้วย compiler ตัวไหน

หนอน Morris · 2 พฤศจิกายน 1988

อินเทอร์เน็ตปี 1988 มีเครื่องต่ออยู่ราว 60,000 เครื่อง และทุกเครื่องใช้ข้อสมมติเดียวกันข้อหนึ่งโดยไม่มีใครเขียนมันลงไปที่ไหน ว่าเครื่องอีกฝั่งของสายไม่ใช่ศัตรู ข้อสมมตินั้นไม่ได้อยู่ในเอกสาร มันอยู่ในบริการเครือข่ายที่ทุกผู้ขายส่งมาพร้อมระบบ

ค่ำวันที่ 2 พฤศจิกายน 1988 โปรแกรมตัวเดียวทำให้เครื่องราว 6,000 จากราว 60,000 เครื่องนั้น — ราวสิบเปอร์เซ็นต์ — หยุดทำงานภายในยี่สิบสี่ชั่วโมง ผู้ปล่อยคือ Robert Tappan Morris นักศึกษาปริญญาโทของ Cornell จากเครื่องที่ MIT

มันไม่ใช่การขโมยข้อมูล มันคือ denial of service โดยไม่ตั้งใจ เครื่องที่ถูกติดเชื้อซ้ำหลายรอบรับภาระจนหยุดเดิน ทางเข้ามีสองทางและทั้งสองทางเคยถูกเรียกว่าคุณสมบัติ ฟังก์ชัน debug ที่ถูกเปิดทิ้งไว้ในระบบจริง กับรูทีนรับข้อมูลที่ไม่จำกัดความยาว แยกกันมันคือความสะดวกสองอย่าง รวมกันมันคือการสั่งรันโค้ดจากระยะไกลบนอินเทอร์เน็ตเกือบทั้งหมด

ตัวเลขความเสียหายมีสองแบบ และแบบที่ตรงกว่าคือแบบที่ดูเหมือนตอบไม่ได้ GAO ประเมินค่าฟื้นฟูไว้ระหว่าง 100,000 ถึง 10,000,000 ดอลลาร์ ช่วงกว้างร้อยเท่า และช่วงนั้นเองคือผลการวัดที่ตรงที่สุด เพราะไม่มีใครวัดได้ ส่วนตัวเลข 100 ล้านดอลลาร์

ที่เล่าต่อกันมาจนติดปาก ไม่มีแหล่งต้นทาง มันดูเหมือนเพดานของ GAO ที่ถูกเติมศูนย์อีกสองตัว เล่มนี้จึงไม่พิมพ์มันเป็นข้อเท็จจริง

สิ่งที่ตามมาหลังคืนนั้นมีสองอย่างที่อยู่คนละระนาบกัน อย่างแรกคือกฎหมาย Morris เป็นคนแรกที่ถูกตัดสินว่าผิดตาม Computer Fraud and Abuse Act ปี 1986 วันที่ 4 พฤษภาคม 1990 ผู้พิพากษา Howard G. Munson ให้รอลงอาญาสามปี บริการชุมชน 400 ชั่วโมง ปรับ 10,050 ดอลลาร์ ไม่มีโทษจำคุก ศาลอุทธรณ์ยืนคำตัดสินในปี 1991

อย่างที่สองคือสถาบัน และมันเกิดขึ้นภายในไม่กี่สัปดาห์ CERT/CC ที่ Software Engineering Institute ของ Carnegie Mellon ด้วยทุนจาก DARPA มันเป็นจุดประสานงานกลางแห่งแรกสำหรับเหตุฉุกเฉินบนเครือข่าย เหตุผลที่ต้องมีเรียบง่ายกว่าที่ควรจะเป็น คืนนั้นไม่มีใครมีเบอร์โทรให้โทรหา

ข้อบกพร่องสองข้ออยู่ในโค้ดที่ส่งมอบแล้วมาหลายปี ผู้ปฏิบัติงานรู้จักมันในฐานะความสะดวก ไม่ใช่ช่องโหว่ ตัวเลขปีที่แน่นอนของแต่ละข้อ การค้นคว้าครั้งนี้ยืนยันไม่ได้ ในแง่สันของเล่ม ข้อสมมติว่าปลายสายเป็นมิตรคือส่วนเผื่อที่ไม่เคยถูกวัดกับใครเลย จนกระทั่งมีคนวัด

สิ่งที่กรณีนี้สอน: ไล่รายการความสะดวกที่ยังเปิดอยู่ในระบบจริง — โหมด debug บัญชีทดสอบ ช่องรับข้อมูลที่ไม่จำกัดขนาด — แล้วถามทีละข้อว่าถ้าปลายสายเป็นศัตรู มันคืออะไร

Phrack 49 · 1996

ก่อนเดือนพฤศจิกายน 1996 โปรแกรมที่พังเพราะรับข้อมูลยาวเกินเป็นเรื่องน่ารำคาญ ไม่ใช่เรื่องความมั่นคง การเขียนเลย stack เป็นเรื่องเล่าในวงเล็กๆ และเป็นบั๊กความไม่เสถียรสำหรับคนที่เหลือ คนเขียนโปรแกรมส่วนใหญ่รู้ว่ามันทำให้โปรแกรมพัง และหยุดคิดตรงนั้น

วันที่ 8 พฤศจิกายน 1996 Phrack ฉบับที่ 49 ไฟล์ที่ 14 จาก 16 พิมพ์บทความของ Aleph One (Elias Levy) และจำนวนผู้เสียหายจากบทความนั้นคือศูนย์ มันเป็นเอกสาร

สิ่งที่บทความตั้งชื่อคือประเภทของข้อบกพร่อง และการตั้งชื่อคือทั้งหมดของเรื่อง โปรแกรมที่เขียนเลยขอบของ buffer ทำลายบันทึกที่บอกว่าจะกลับไปทำงานต่อที่ไหนได้ ถ้าบันทึกนั้นถูกทำลายได้ มันก็ถูกเลือกค่าให้ได้ ประเภทของข้อบกพร่องจึงไม่ใช่โปรแกรมพัง แต่คือผู้โจมตีเลือกได้ว่าอะไรจะรันต่อ เล่มนี้บันทึกแค่นั้น ไม่มีเทคนิค ไม่มีโค้ด โดยตั้งใจ

หลังบทความ มันเป็นประเภทที่มีชื่อ สอนได้ ทำซ้ำได้ มีคำศัพท์มาตรฐาน สอนผู้โจมตีได้ และที่ส่งผลมากกว่า สอนผู้ป้องกันได้ แนวป้องกันที่ตามมาเป็นเส้นตรง StackGuard canary ปี 1998 ASLR ของ PaX ปี 2001 Windows /GS ปี 2003 บิต NX / DEP ในฮาร์ดแวร์ปี 2004 สังเกตว่าทุกชิ้นตอบบทความ ไม่ได้ตอบเหตุการณ์ ไม่มีภัยพิบัติไหนอยู่ระหว่างปี 1996 กับ StackGuard มีแต่คนอ่านหนังสือ

ระยะเวลาที่ข้อบกพร่องมีอยู่ก่อนถูกใช้ กลับด้านในกรณีนี้ ประเภทนี้ถูกใช้จริงแปดปีก่อนบทความ — ทางเข้าทางหนึ่งของหนอน Morris ปี 1988 คือมัน ความเชื่อว่าบทความนี้สร้าง buffer overflow ขึ้นมา จึงตายในการค้นคว้า ปี 1996 ไม่ได้ทำให้มันมีอยู่ มันทำให้ทุกคนได้อ่าน และข้ออ้างนั้นใหญ่กว่าคำว่าประดิษฐ์

มีข้อเทียบที่ไม่สบายใจอยู่ข้อหนึ่ง Karger และ Schell บันทึกในบทย้อนหลังปี 2002 ว่า Multics ต้านข้อบกพร่องประเภทนี้ทั้งประเภทได้ตั้งแต่ปี 1974 เพราะบิตสิทธิ์ execute ในฮาร์ดแวร์ทำให้หน้าข้อมูลถูกรันเป็นคำสั่งไม่ได้ การป้องกันมาก่อนการแพร่กระจายยี่สิบสองปี แล้วหายไปจากระบบที่ต้องการมัน เหตุผลไม่ใช่ว่าใครลืม แต่เพราะระบบเหล่านั้นเลือกสถาปัตยกรรมฮาร์ดแวร์ที่พูดเรื่องนี้ไม่ได้ ของที่จะกลับมาได้ต้องรอฮาร์ดแวร์รุ่นที่พูดได้ และมันมาถึงปี 2004

ในแง่สันของเล่ม บทความคือวันที่ผู้โจมตีทุกคนได้อ่านแบบก่อสร้างชุดเดียวกัน ส่วนเผื่อที่เคยเป็นความไม่รู้ หมดอายุในวันนั้น

สิ่งที่กรณีนี้สอน: เมื่อบั๊กประเภทหนึ่งได้ชื่อและมีบทความสอน ให้นับวันนั้นเป็นวันแรกที่ระบบของคุณถูกอ่าน แล้วไล่ดูว่าการป้องกันที่ตามมาแต่ละชิ้นเปิดอยู่บนเครื่องของคุณหรือไม่

SQL injection และ Heartland · 1998 และ 2008

หน้าเว็บที่รับข้อความจากผู้ใช้แล้วส่งต่อให้ฐานข้อมูล เป็นของธรรมดามาตั้งแต่เว็บเริ่มทำอะไรได้มากกว่าแสดงเอกสาร และคนที่เขียนมันคิดว่าข้อความที่ผู้ใช้พิมพ์คือค่าหนึ่งค่าในคำสั่ง วันที่ 25 ธันวาคม 1998 Phrack ฉบับที่ 54 ไฟล์ที่ 8 พิมพ์บทความของ rain.forest.puppy (Jeff Forristal) ที่อธิบายว่ามันไม่ใช่ ข้อความจากผู้ใช้ที่ไปถึงฐานข้อมูลโดยไม่ถูกกรอง เปลี่ยนโครงสร้างของคำสั่งได้ ไม่ใช่แค่ค่าในคำสั่ง เก้าปีต่อมา ข้อบกพร่องประเภทเดียวกันเปิดทางให้บันทึกบัตร 130 ล้านรายการถูกเข้าถึง

Forristal ไม่ได้ใช้คำว่า SQL injection ชื่อนั้นมาทีหลัง สิ่งที่เขาบรรยายคือหน้าเว็บบน ASP ที่ต่อกับ SQL Server 6.5 และรับข้อความจากผู้ใช้ไปต่อเข้าคำสั่งฐานข้อมูลตรงๆ ตั้งแต่วันคริสต์มาสนั้น ประเภทของข้อบกพร่องนี้เป็นสาธารณะ ใครก็อ่านได้ ทั้งคนที่ต้องปิดมันและคนที่ต้องการใช้มัน

Heartland Payment Systems เป็นผู้ประมวลผลการชำระเงินใหญ่อันดับห้าของสหรัฐ ตามคำฟ้องของกระทรวงยุติธรรม การบุกรุกเครือข่ายเริ่มอย่างช้าที่สุดวันที่ 26 ธันวาคม 2007 Visa แจ้งเตือน Heartland เรื่องความเคลื่อนไหวผิดปกติของบัตร ปลายเดือนตุลาคม 2008 บริษัทประกาศต่อสาธารณะวันที่ 20 มกราคม 2009 ราวสิบสามเดือนโดยไม่มีใครเห็น และคนที่เห็นก่อนบริษัทคือ Visa

ทางเข้าคือ SQL injection สู่เครือข่ายสำนักงาน การเก็บเกี่ยวทำด้วยตัวดักฟังที่วางไว้บนเครือข่ายประมวลผล และมันทำงานอยู่หลายเดือน ระหว่างสองขั้นนั้นมีข้อสมมติหนึ่งข้อที่ไม่มีใครตรวจ ต้นเหตุในหนึ่งประโยค แอปพลิเคชันหน้าเว็บต่อข้อความที่ไม่น่าไว้ใจเข้าคำสั่งฐานข้อมูล และเครือข่ายสำนักงานที่มันอยู่ไม่ได้แยกจากเครือข่ายที่ถือข้อมูลบัตรจริง วันที่ SQL injection สำเร็จครั้งแรก แยกจากวันเข้าเครือข่ายในคำฟ้อง การค้นคว้าครั้งนี้หาไม่พบ

ตัวเลข 130 ล้านคือจำนวนบันทึกบัตรที่ถูกเข้าถึงตามคำฟ้อง ไม่ใช่จำนวนบัตรที่ถูกใช้ทุจริต รายงานร่วมสมัยบางฉบับใช้ "ถึง 100 ล้าน" ตัวเลขแข็งอยู่ในรายงานต่อ ก.ล.ต. ของบริษัทเอง เพราะบริษัทโกหกตัวเลขในรายงานนั้นไม่ได้ Heartland บันทึกค่าใช้จ่าย 146.4 ล้านดอลลาร์ระหว่างมกราคม 2009 ถึงมีนาคม 2011 ได้คืนจากประกัน 31.2 ล้าน ราว 114.7 ล้านเป็นเงินยอมความ Visa 58.6 ล้าน จ่ายวันที่ 18 กุมภาพันธ์

2010 MasterCard 34.4 ล้าน เดือนกันยายน 2010 American Express ราว 3.5 ล้าน Albert Gonzalez ถูกฟ้องเดือนสิงหาคม 2009 และรับโทษจำคุกยี่สิบปี

เก้าปี ธันวาคม 1998 ถึงธันวาคม 2007 ระหว่างการตีพิมพ์ประเภทของข้อบกพร่องกับการรั่วของบัตรครั้งใหญ่ที่สุดที่เคยบันทึกในเวลานั้น ผู้โจมตีอ่านบทความฉบับเดียวกับผู้ป้องกัน ต่างกันที่ฝ่ายหนึ่งลงมือ

สิ่งที่กรณีนี้สอน: เขียนปีที่ข้อบกพร่องประเภทนั้นถูกตีพิมพ์ครั้งแรกไว้ข้างช่องโหว่ทุกช่องที่กำลังปิด — ตัวเลขนั้นคือจำนวนปีที่ผู้โจมตีมีมากกว่าคุณ

Stuxnet · 2010

ในช่วงพฤศจิกายน 2009 ถึงปลายมกราคม 2010 เครื่องหมุนเหวี่ยง IR-1 ถึงราวหนึ่งพันเครื่องที่โรงงานเสริมสมรรถนะ Natanz ถูกเปลี่ยนออก ตลอดช่วงเวลานั้น จอของผู้ควบคุมอ่านค่าปกติ และมันอ่านค่าปกติเพราะมีคนตั้งใจให้มันอ่านอย่างนั้น

สิ่งที่ในที่สุดทำให้เรื่องนี้ถูกพบไม่ใช่หน่วยข่าวกรองไหน วันที่ 17 มิถุนายน 2010 บริษัทแอนติไวรัสเล็กๆ ในมินสค์ชื่อ VirusBlokAda พบมัลแวร์ตัวหนึ่งจากคำร้องเรียนของลูกค้าเรื่องเครื่องรีบูตเองโดยไม่มีสาเหตุ นักวิเคราะห์ที่พบคือ Sergey Ulasen ปฏิบัติการเดินมาอย่างน้อยตั้งแต่ปี 2009

ตัวเลขหนึ่งพันมาจากสถาบัน ISIS เดือนธันวาคม 2010 คำที่สถาบันใช้คือเสียหายและถูกเปลี่ยน ราวสิบเปอร์เซ็นต์ของเครื่องที่เดินอยู่ คำว่าถูกทำลายเป็นคำที่นิยมเล่าและอาจเกินแหล่งต้นทาง รายงานของ Symantec สืบต้นทางการติดเชื้อไปถึงห้าองค์กร ทุกแห่งเกี่ยวข้องกับโครงการนิวเคลียร์ของอิหร่าน

มันใช้ zero-day ของ Windows สี่ตัว ช่องโหว่ในไฟล์ shortcut ในระบบพิมพ์ และช่องยกระดับสิทธิ์อีกสอง ตัวเลขห้าที่เล่าต่อกันมาผิด และความผิดของมันไม่ใช่เรื่องเลขคลาดเคลื่อนหนึ่งหน่วย ช่องที่ห้าคือ MS08-067 ที่มีแพตช์มาตั้งแต่ปี 2008 และถูกเลือกเพราะมันเก่า ด้วยเหตุผลว่าเครือข่ายควบคุมอุตสาหกรรมแพตช์ช้า บันทึกน้อย และป้องกันเบา พูดอีกอย่างคือผู้เขียนมัลแวร์งบประมาณช่องโหว่ที่รู้กันแล้วไว้

ใส่จุดอ่อนของสถาบันที่รู้กันแล้ว นั่นคือข้อเท็จจริงที่น่าสนใจที่สุดของปฏิบัติการ และคำว่าห้า zero-day ทำให้มันหาย

เครือข่ายเป้าหมายไม่มีเส้นทางจากอินเทอร์เน็ต หนอนจึงเดินทางบนสื่อพกพา ผ่านเครื่องของวิศวกร และผ่านไฟล์โครงการที่เครื่องเหล่านั้นถือเข้าไปข้างใน มันขี่การจราจรซ่อมบำรุงธรรมดาที่ air gap ต้องมี เพื่อให้ air gap ดูแลรักษาได้ ช่องว่างไม่ได้ถูกเอาชนะ มันถูกใช้ ขั้นตอนปฏิบัติของ air gap เองคือพาหนะ ต้นเหตุในหนึ่งประโยค: การแยกระบบที่อนุญาตให้ข้ามได้ทางเดียวคือคนถือสื่อเข้าไป คือการแยกระบบที่มีช่องทางผ่านซึ่งเขียนเป็นเอกสาร มีตารางเวลา และไม่ตรวจตัวตน

สิ่งที่ไม่เคยมีมาก่อนคือสองอย่างพร้อมกัน มันเป็นมัลแวร์ตัวแรกที่เขียนเพื่อทำลายอุปกรณ์อุตสาหกรรมทางกายภาพ ไม่ใช่เพื่อขโมย ปิดกั้น หรือขีดเขียน และมันเล็ง PLC ตระกูล Siemens S7 เปลี่ยนกระบวนการ แล้วเล่นค่าปกติที่บันทึกไว้ก่อนหน้ากลับไปที่จอของผู้ควบคุม สองอย่างนี้ต้องมาด้วยกัน เพราะการทำลายเครื่องจักรช้าๆ ต้องการเวลา และเวลาจะมีได้ก็ต่อเมื่อคนที่เฝ้าอยู่ไม่รู้ตัว เครื่องวัดอ่านว่าปกติ ขณะที่เครื่องจักรพังตัวเอง ในเล่มที่ถามว่าเผื่อพอหรือไม่ นี่คือกรณีที่เครื่องวัดส่วนเผื่อถูกยึดก่อน

สิ่งที่กรณีนี้สอน: เขียนรายการทางข้ามทุกทางของ air gap — แผ่นสื่อ เครื่องของช่าง ไฟล์โครงการ — แล้วถามว่าทางไหนตรวจสิ่งที่ข้ามมา และเครื่องวัดที่คุณเชื่อ อ่านค่าจากคอมพิวเตอร์ตัวเดียวกับที่ถูกโจมตีหรือไม่

Heartbleed · เมษายน 2014

การเข้ารหัสของอินเทอร์เน็ตส่วนใหญ่วิ่งผ่านไลบรารีตัวเดียวชื่อ OpenSSL และไลบรารีตัวนั้นดูแลด้วยเงินบริจาคราว 2,000 ดอลลาร์ต่อปี ตัวเลขสองตัวนี้อยู่ในโลกเดียวกันมาหลายปีโดยไม่มีใครต้องอธิบายมัน จนกระทั่งวันที่ 7 เมษายน 2014

วันนั้น OpenSSL ออกประกาศช่องโหว่ CVE-2014-0160 และในเดือนเดียวกัน Netcraft นับได้ว่าเว็บไซต์ SSL 17.5% เปิดส่วนขยายที่มีช่องโหว่นี้อยู่ — ใบรับรองราวครึ่งล้านใบจากผู้ออกที่เบราว์เซอร์เชื่อถือ ผู้พบคือ Neel Mehta จาก Google Security และบริษัท Codenomicon แยกกันพบ แก้ใน OpenSSL 1.0.1g

กลไกในระดับแนวคิดเรียบง่ายจนน่าหงุดหงิด ส่วนขยาย heartbeat ของ TLS ตาม RFC 6520 ให้ฝ่ายหนึ่งส่งข้อมูลก้อนเล็กพร้อมระบุความยาว แล้วขอให้อีกฝ่ายส่งกลับ OpenSSL ใช้ความยาวที่ถูกบอก แทนที่จะวัดก้อนที่ได้รับจริง เมื่อความยาวที่บอกเกินของจริง คำตอบถูกเติมให้ครบขนาดด้วยสิ่งที่อยู่ถัดไปในหน่วยความจำของโปรเซส ได้ครั้งละถึง 64 KB ทำซ้ำได้ ไม่ต้องมีรหัสผ่าน ไม่ต้องเปิดเซสชันให้เสร็จ และไม่ทิ้งอะไรไว้ในบันทึก ส่วนสิ่งที่อยู่ถัดไปบนเซิร์ฟเวอร์ TLS รวมถึงข้อมูลเซสชันและกุญแจส่วนตัว

ระยะเวลาของกรณีนี้ชัดที่สุดในภาค และมันน่าสนใจตรงที่ใครเป็นคนเขียน โค้ดที่ผิดถูกเพิ่มวันที่ 31 ธันวาคม 2011 โดย Robin Seggelmann ผู้เขียนร่วมคนแรกของ RFC ฉบับนั้นเอง ส่งถึงโลกใน OpenSSL 1.0.1 วันที่ 14 มีนาคม 2012 เปิดเผยวันที่ 7 เมษายน 2014 สองปีกับสามสัปดาห์ในโค้ดที่ปล่อยใช้แล้ว ระหว่างสองปีนั้น Apache กับ nginx สองเซิร์ฟเวอร์ที่พึ่ง OpenSSL มากที่สุด มีส่วนแบ่งเว็บไซต์ที่ใช้งานอยู่รวมกันเกิน 66% ในสำรวจของ Netcraft เดือนเดียวกัน ต้นเหตุในหนึ่งประโยค: ค่าความยาวที่ฝ่ายไกลส่งมาถูกเชื่อแทนที่จะถูกตรวจ ในไลบรารีความมั่นคงที่ถูกติดตั้งกว้างที่สุดที่มีอยู่

ส่วนเผื่อถูกเติมหลังจากมันหมดไปแล้ว วันที่ 24 เมษายน 2014 สิบเจ็ดวันหลังเปิดเผย Linux Foundation ประกาศ Core Infrastructure Initiative โดยมีบริษัทเทคโนโลยีใหญ่หลายแห่งหนุน OpenSSL เป็นโครงการชุดแรกที่ได้ทุน และเงินนั้นจ่ายเป็นเงินเดือนนักพัฒนาแกนหลักเต็มเวลาสองคน สองคน หลังจากสองปีสามสัปดาห์

ในแง่สันของเล่ม ธรรมชาติไม่ส่งค่าความยาวปลอม ผู้โจมตีส่ง และส่งซ้ำได้ไม่จำกัด ค่าที่ไม่ถูกตรวจในโค้ดหนึ่งบรรทัดจึงไม่ใช่บั๊กหนึ่งตัว มันคือประตูที่เปิดอยู่บนเซิร์ฟเวอร์ครึ่งล้านเครื่องพร้อมกัน

สิ่งที่กรณีนี้สอน: หาไลบรารีที่ทุกอย่างของคุณพึ่งอยู่ แล้วถามว่ามีคนกี่คนได้เงินเดือนดูแลมัน — ถ้าคำตอบคือศูนย์ นั่นคือส่วนเผื่อที่ไม่มี

Mirai · ตุลาคม 2016

เจ้าของเราเตอร์ตามบ้าน กล้อง IP และเครื่องบันทึกวิดีโอ ไม่รู้ว่าของพวกนั้นเป็นคอมพิวเตอร์ พวกเขาซื้อมันมาเพื่อทำงานอย่างเดียว เสียบไฟ แล้วลืม ผู้ผลิตเผารหัสผ่านตายตัวลงในเฟิร์มแวร์ก่อนส่งออกจากโรงงาน และหลายกรณีเจ้าของเปลี่ยนมันไม่ได้ต่อให้อยากเปลี่ยน

วันที่ 21 ตุลาคม 2016 ผู้ให้บริการ DNS ชื่อ Dyn ถูกโจมตีจากปลายทางถึงราวหนึ่งแสนจุดตามการวิเคราะห์ของ Dyn เอง และ Twitter, Netflix, Reddit, GitHub กับ Spotify หายไปจากผู้ใช้ในพื้นที่ส่วนใหญ่ของสหรัฐ

ลำดับเหตุการณ์ก่อนหน้านั้นสั้น วันที่ 20 กันยายน เว็บไซต์ Krebs on Security ถูกโจมตี วันที่ 30 กันยายน ผู้ใช้ชื่อ Anna-senpai ปล่อย source code ของมัลแวร์สู่สาธารณะ วันที่ 21 ตุลาคม Dyn ล้ม สามสัปดาห์ระหว่างการแจกเครื่องมือกับการใช้มันในระดับนั้น ตัวสแกนของ Mirai ถือพจนานุกรมชื่อผู้ใช้กับรหัสผ่าน 62 คู่ ไม่ซ้ำกัน 61 คู่ มีหนึ่งคู่ซ้ำ นั่นคือเหตุผลที่วรรณกรรมพิมพ์ทั้งสองตัวเลข และหกสิบสองคู่คือทั้งหมดที่มันต้องใช้ มันไม่ใช่เครื่องมือถอดรหัส มันเป็นแค่รายชื่อของที่ผู้ผลิตพิมพ์ไว้ในคู่มือ

งานศึกษาใน USENIX Security 2017 โดย Antonakakis และคณะ ติดตามเจ็ดเดือน วัดจุดสูงสุดของการติดเชื้อพร้อมกันได้ 600,000 เครื่อง ส่วนใหญ่เป็นเราเตอร์ตามบ้าน กล้อง IP และเครื่องบันทึกวิดีโอ ส่วนตัวเลขสะสมสองถึงสามล้านเครื่องที่แพร่อยู่การค้นคว้าครั้งนี้ไม่ได้ตรวจ

ตัวเลข 1.2 Tbps ที่ถูกพิมพ์เป็นข้อเท็จจริงมาสิบปี ตายในการค้นคว้า และคนที่ฆ่ามันคือ Dyn เอง บริษัทเขียนว่ามีรายงานขนาดในช่วงนั้น และในเวลานั้นบริษัทยืนยันข้ออ้างนี้ไม่ได้ Dyn ยังบอกด้วยว่าตัวเลขของตนไม่รวมการจราจรที่ถูกกรองต้นทาง ตัวเลขจริงจึงไม่รู้ว่ามากหรือน้อยกว่า มันแค่ไม่รู้ และการที่ตัวเลขนี้หายไปไม่ได้ทำให้เรื่องเล็กลง เพราะสิ่งที่รู้คือคานงัดไม่ใช่แบนด์วิดท์ การแปลงชื่อเป็นที่อยู่คือจุดล้มจุดเดียวของทุกอย่างที่อยู่เหนือมัน ล้ม Dyn ได้ก็ล้มทุกชื่อที่ Dyn ถืออยู่

ต้นเหตุในหนึ่งประโยค: อุปกรณ์ถูกส่งออกจากโรงงานพร้อมรหัสผ่านตายตัวที่ตีพิมพ์แล้ว บนบริการที่เข้าถึงได้จากอินเทอร์เน็ตเปิด ไม่มีกลไกอัปเดต และเจ้าของไม่รู้ว่าสิ่งนั้นเป็นคอมพิวเตอร์

คำถามประจำภาค — ข้อบกพร่องมีอยู่นานเท่าไรก่อนถูกใช้ — ไม่มีคำตอบในกรณีนี้ และเหตุผลที่ไม่มีคือเหตุผลที่แย่ที่สุด มันไม่ใช่ข้อบกพร่องที่เกิดในวันหนึ่งและแก้ด้วยแพตช์ มันคือการตัดสินใจในการผลิต อยู่ในฮาร์ดแวร์ที่มีอายุใช้งานหลายปีและไม่มีทางส่งการแก้ไขไปถึง ข้อบกพร่องคือตัวสินค้า

สิ่งที่กรณีนี้สอน: ก่อนต่ออุปกรณ์ใดเข้าเครือข่าย ถามสองข้อ — เปลี่ยนรหัสผ่านจากโรงงานได้ไหม และผู้ผลิตส่งแพตช์มาทางไหน — ถ้าตอบไม่ได้ทั้งคู่ ให้ถือว่ามันติดเชื้อแล้วตั้งแต่วันซื้อ

Equifax · 2017

วันที่ 7 มีนาคม 2017 Apache ประกาศช่องโหว่ CVE-2017-5638 ใน Struts พร้อมแพตช์ในวันเดียวกัน แบบก่อสร้างของการโจมตีจึงวางอยู่ในที่สาธารณะพร้อมคำเตือนและทางแก้ตั้งแต่นาทีแรก และ 67 วันต่อมา ผู้โจมตีเดินเข้าพอร์ทัลรับเรื่องร้องเรียนของ Equifax ผ่านช่องโหว่นั้นที่ยังไม่ได้แพตช์

เรื่องจริงของกรณีนี้ไม่ได้อยู่ที่แพตช์ที่พลาด มันอยู่ที่ทำไมไม่มีใครเห็นอะไรเลยตลอดสองเดือนครึ่งหลังจากนั้น อุปกรณ์ที่ตรวจการจราจรเข้ารหัสในเครือข่ายของ Equifax หยุดทำงานมา 19 เดือน เพราะใบรับรองของมันหมดอายุ วันที่ 29 กรกฎาคม พนักงานคนหนึ่งต่ออายุใบรับรอง เครื่องมือเริ่มแจ้งเตือนการจราจรผิดปกติทันที ความสามารถนี้ถูกซื้อ ถูกติดตั้ง และตาบอด และไม่มีอะไรในองค์กรสังเกตว่ากลไกควบคุมชิ้นหนึ่งหยุดส่งผลลัพธ์มาปีครึ่ง

วันที่เรียงตามลำดับที่สำคัญ เข้าระบบ 13 พฤษภาคม ระหว่างพฤษภาคมถึงปลายกรกฎาคม ผู้โจมตีค้นฐานข้อมูลของ Equifax แยกกัน 51 ฐาน ตรวจพบ 29 กรกฎาคม ซึ่งเป็นวันเดียวกับที่ใบรับรองถูกต่ออายุ เปิดเผยต่อสาธารณะ 7 กันยายน ระยะเวลาที่ผู้โจมตีอยู่ในระบบ ตัวเลขที่ใช้กันทั่วไปคือ 76 วัน นับตามปฏิทินจาก 13 พฤษภาคม ถึง 29 กรกฎาคมได้ 77 วัน เล่มนี้พิมพ์ทั้งสอง

ขนาด 147 ล้านคน ตามตัวเลขของ Equifax และ FTC ราวครึ่งหนึ่งของประชากรสหรัฐ รวมผู้อยู่อาศัยในสหราชอาณาจักรและแคนาดา ตัวเลขของ GAO คืออย่างน้อย 145.5 ล้านคน นับเฉพาะผู้บริโภคในสหรัฐ ทั้งสองตัวเลขถูกต้อง มันนับคนละประชากร ต้นเหตุในหนึ่งประโยค: ช่องโหว่ที่มีแพตช์สาธารณะถูกปล่อยให้เดินอยู่ 67 วันบนแอปพลิเคชันที่หันหน้าหาอินเทอร์เน็ต ในเครือข่ายที่ไม่แบ่งส่วน ภายใต้ระบบเฝ้าระวังที่ถูกปิดด้วยใบรับรองหมดอายุและไม่มีใครคิดถึง

ค่ายอมความราว 700 ล้านดอลลาร์ กับ FTC, CFPB และ 50 รัฐ เดือนกรกฎาคม 2019 แบ่งเป็น 575 ล้านทันที และเพิ่มได้อีกถึง 125 ล้าน รายงานชันสูตรของ GAO หมายเลข GAO-18-559 เดือนสิงหาคม 2018 ระบุความบกพร่องเชิงระบบสี่ด้าน การระบุ การตรวจจับ การแบ่งส่วน และการกำกับข้อมูล สี่ความล้มเหลว ไม่ใช่แพตช์ที่พลาดหนึ่งตัว

ในแง่สันของเล่ม 67 วันคือระยะที่แบบก่อสร้างพร้อมคำเตือนวางอยู่ในที่สาธารณะและผู้โจมตีอ่านมันก่อนเจ้าของ

สิ่งที่กรณีนี้สอน: ให้เครื่องมือตรวจจับทุกตัวรายงานจำนวนสิ่งที่มันเห็นต่อวัน แล้วตั้งสัญญาณเตือนเมื่อตัวเลขนั้นเป็นศูนย์ — ความเงียบของเครื่องมือไม่ใช่ความสงบของระบบ

NotPetya · มิถุนายน 2017

ใครทำธุรกิจในยูเครนปี 2017 แทบต้องใช้ซอฟต์แวร์บัญชีภาษีชื่อ M.E.Doc ยื่นภาษี ฐานผู้ติดตั้งจึงคือบริษัทที่ทำธุรกิจในยูเครน รวมถึงสำนักงานท้องถิ่นของบริษัทข้ามชาติ และทุกเครื่องที่ติดตั้งมันถูกตั้งค่าให้รับอัปเดตจากเซิร์ฟเวอร์ของ M.E.Doc เอง โดยอัตโนมัติ เพราะนั่นคือสิ่งที่ซอฟต์แวร์ที่ดูแลดีควรทำ

วันที่ 27 มิถุนายน 2017 หนึ่งวันก่อนวันรัฐธรรมนูญของยูเครน Maersk บริษัทเดินเรือคอนเทนเนอร์ใหญ่ที่สุดในโลกสูญเสีย domain controller ทุกตัวในเครือข่ายทั่วโลก และต้องติดตั้ง PC ราว 45,000 เครื่องกับเซิร์ฟเวอร์ 4,000 เครื่องใหม่ทั้งหมด

ทางที่มันเข้าไปถึงที่นั่นคือเซิร์ฟเวอร์อัปเดต ผู้โจมตียึดเซิร์ฟเวอร์อัปเดตของ M.E.Doc เปลี่ยนเส้นทางคำขออัปเดตจากเครื่องลูกค้าไปยังเซิร์ฟเวอร์ของตน และสั่งให้ดาวน์โหลดและรัน ทุกเหยื่อรับมันผ่านช่องทางที่ตนตั้งใจตั้งค่าให้ไว้ใจและรันอัตโนมัติ ต้นเหตุในหนึ่งประโยค: ช่องทางอัปเดตตรวจตัวตนของเซิร์ฟเวอร์ ไม่ได้ตรวจตัวสิ่งที่ส่งมา ยึดเซิร์ฟเวอร์ได้จึงเท่ากับยึดทุกเครื่องปลายทาง

มันไม่ใช่ ransomware มันแสดงข้อความเรียกค่าไถ่ แต่กุญแจติดตั้งที่แสดงเป็นค่าสุ่ม จ่ายแล้วถอดรหัสไม่ได้ มันคือตัวลบข้อมูลที่สวมเสื้อ ransomware ข้อนี้เปลี่ยนความหมายของทุกเรื่องเล่าที่ตามมา เรื่องเล่าการฟื้นตัวทุกเรื่องจึงเป็นการสร้างใหม่ ไม่ใช่การถอดรหัส

domain controller ทุกตัวของ Maersk ถูกทำลาย และเครือข่าย Windows ที่ไม่มี Active Directory สร้างใหม่ไม่ได้ เพราะไม่เหลืออะไรบอกว่าใครเป็นใคร เหลืออยู่ตัวเดียว เครื่องในสำนักงานห่างไกลที่กานา ไฟดับในพื้นที่ทำให้มันหลุดจากเครือข่ายก่อน NotPetya ไปถึง การฟื้นตัวทั้งหมดของบริษัทเดินเรือใหญ่ที่สุดในโลกขึ้นอยู่กับอุบัติเหตุในสำนักงานที่มีทรัพยากรน้อยที่สุด

ตัวเลขความเสียหายมีสองชั้น และเล่มนี้พิมพ์ทั้งสองชั้นเพราะทั้งสองชั้นบอกคนละเรื่อง ทำเนียบขาววันที่ 15 กุมภาพันธ์ 2018 ระบุว่าเป็นฝีมือกองทัพรัสเซีย เรียกมันว่าการโจมตีไซเบอร์ที่ทำลายล้างและแพงที่สุดในประวัติศาสตร์ และให้ตัวเลขเกินหนึ่งหมื่นล้านดอลลาร์ ตัวเลขนั้นไม่มีฐานการตรวจสอบและไม่มีวิธีคำนวณตีพิมพ์ มันจึงอยู่ในเล่มนี้ในฐานะคำแถลงของทำเนียบขาว ไม่ใช่ในฐานะจำนวนเงิน ส่วนตัวเลขที่บริษัทเปิดเผยเองตรวจได้ Merck ราว 870 ล้าน FedEx/TNT Express 300 ล้านในไตรมาสเดียวและเกิน 400 ล้านรวม Maersk 250 ถึง 300 ล้าน รวมแล้วต่ำกว่าสองพันล้านที่ระบุที่มาได้ ส่วนหางยาวที่เหลือไม่รู้จริง

คำถามประจำภาคตอบได้สองชั้นเช่นกัน ส่วนที่ใช้แพร่ในเครือข่ายอาศัยช่องโหว่ SMB ที่ MS17-010 แพตช์ไว้วันที่ 14 มีนาคม 2017 สามเดือนครึ่งก่อนหน้า แต่ทางเข้าหลักคือช่องทางอัปเดตเอง และนั่นไม่ใช่ข้อบกพร่องที่มีอายุ มันคือแบบที่ออกไว้

สิ่งที่กรณีนี้สอน: หาช่องทางทุกช่องที่เครื่องของคุณรับโค้ดมารันโดยอัตโนมัติ แล้วถามว่ามันตรวจลายเซ็นของสิ่งที่ส่งมา หรือแค่ตรวจว่าเซิร์ฟเวอร์ถูกตัว

SolarWinds · ธันวาคม 2020

องค์กรราว 300,000 แห่งเป็นลูกค้าของ SolarWinds และงานประจำที่ไม่มีใครต้องคิดมากคือติดตั้งอัปเดตของซอฟต์แวร์ Orion ที่บริษัทลงลายเซ็นส่งมา ลายเซ็นคือสิ่งที่ทำให้ไม่ต้องคิด มันตอบคำถามว่าของนี้มาจากใคร

วันที่ 13 ธันวาคม 2020 FireEye เปิดเผยว่าอัปเดตที่ SolarWinds ลงลายเซ็นและแจกจ่ายระหว่างเดือนมีนาคมถึงมิถุนายนปีนั้น มี backdoor อยู่ข้างใน และลูกค้าถึง 18,000 รายติดตั้งมันไปแล้ว FireEye พบมันขณะสอบสวนการถูกเจาะของตัวเอง

สิ่งที่อยู่ในระบบ build ของ SolarWinds คือมัลแวร์อีกตัวชื่อ SUNSPOT และจังหวะที่มันเลือกคือทั้งหมดของเรื่อง มันเฝ้าดูโปรเซสที่แปลโค้ดของ Orion และสลับไฟล์ source หนึ่งไฟล์ในจังหวะ build เพื่อฝัง backdoor ชื่อ SUNBURST หลังนักพัฒนาตรวจโค้ดเสร็จ ก่อนลงลายเซ็น ผลคือคลัง source สะอาด และ binary ที่ลงลายเซ็นแล้วไม่สะอาด การตรวจทุกชั้นปลายทาง ลายเซ็นถูกต้อง ผู้เผยแพร่ตัวจริง ค่า hash ตรงกับที่ SolarWinds ประกาศ ตอบว่าผ่านทั้งหมด ไม่มีชั้นไหนตอบผิด ทุกชั้นตอบคำถามที่ตัวเองถูกสร้างมาให้ถาม นี่คือข้อโต้แย้งของ Thompson ปี 1984 และคำทำนายในมาตรา 3.2.4 ของรายงาน Multics ปี 1974 ทำงานจริงในระดับอุตสาหกรรม ต้นเหตุในหนึ่งประโยค: ระบบ build ถูกไว้ใจและไม่ถูกเฝ้า กุญแจลงลายเซ็นจึงรับรอง binary ที่ไม่มีมนุษย์คนไหนเคยอ่าน

ตัวเลขที่ถูกรายงานผิดบ่อยที่สุดคือ 18,000 นั่นคือจำนวนลูกค้าที่ติดตั้งอัปเดตปนเปื้อน จากลูกค้าราว 300,000 ราย SolarWinds ระบุจำนวนที่ถูกใช้ประโยชน์จริงไว้ต่ำกว่า 100 ทำเนียบขาวโดย Anne Neuberger วันที่ 17 กุมภาพันธ์ 2021 ระบุหน่วยงานรัฐบาลกลางเก้าแห่งและบริษัทเอกชนราว 100 แห่ง ติดตั้งไม่เท่ากับถูกเจาะ

ช่องว่างระหว่าง 18,000 กับราวหนึ่งร้อยคือตัวเลขที่น่าสนใจกว่าทั้งสองตัว ผู้โจมตีสร้างกลไกส่งของที่ไปถึงหนึ่งหมื่นแปดพันองค์กร แล้วใช้มันกับเศษเสี้ยวของหนึ่งเปอร์เซ็นต์ เพราะเปิดใช้ทั้งหมดจะเผามันทิ้ง ความยับยั้งชั่งใจตรงนั้นไม่ใช่ความเมตตา มันคือการถนอมทรัพย์สิน

ระยะเวลา: เข้าถึงระบบของ SolarWinds เองราวเดือนกันยายน 2019 ทดลองฝังเดือนตุลาคม 2019 backdoor ตัวจริงออกไปกับ build ระหว่างมีนาคมถึงมิถุนายน

2020 ถูกพบเดือนธันวาคม 2020 ราวสิบสี่ถึงสิบห้าเดือนจากเข้าครั้งแรกถึงเปิดเผย และมีความพยายามลบร่องรอยจำนวนมากที่ใช้ไปเพื่อยืดตัวเลขนี้โดยเฉพาะ

ในแง่สันของเล่ม ผู้โจมตีไม่ได้เจาะแบบก่อสร้าง เขาเจาะโรงงานที่หล่อชิ้นส่วน แล้วปล่อยให้ระบบตรวจรับของทุกลูกค้าประทับตราให้

สิ่งที่กรณีนี้สอน: สร้าง binary จาก source เดียวกันบนเครื่องที่สอง แล้วเทียบไบต์ต่อไบต์กับสิ่งที่ออกจากระบบ build — ลายเซ็นบอกว่าใครส่ง ไม่ได้บอกว่าข้างในคืออะไร

Log4Shell · ธันวาคม 2021

งานที่น่าเบื่อที่สุดที่ซอฟต์แวร์ทำ คือการจดว่าเกิดอะไรขึ้น และไลบรารีที่รับงานนั้นในโลก Java ชื่อ Log4j มันอยู่แทบทุกที่ที่ Java ทำงาน เพราะแทบทุกโปรแกรมต้องจดอะไรสักอย่าง ปัญหาอยู่ตรงที่มันไม่ได้แค่จด ต้นเหตุในหนึ่งประโยค: ไลบรารีบันทึก log ปฏิบัติต่อข้อความที่ถูกขอให้จด ในฐานะสิ่งที่ต้องประเมิน แทนที่จะเป็นสิ่งที่ต้องเก็บ

วันที่ 24 พฤศจิกายน 2021 Chen Zhaojun จาก Alibaba Cloud Security Team รายงานช่องโหว่ CVE-2021-44228 ใน Log4j ต่อ Apache และเมื่อมันเป็นสาธารณะ วันที่ 9–10 ธันวาคม คะแนน CVSS ของมันคือ 10.0 จาก 10

สิบเต็มแปลว่าสั่งรันโค้ดจากระยะไกลได้โดยไม่ต้องยืนยันตัวตน จุดชนวนง่าย และไลบรารีอยู่แทบทุกที่ที่ Java ทำงาน การแก้ไล่มาสามรุ่น 2.15.0 แล้ว 2.16.0 ที่ปิด JNDI เป็นค่าตั้งต้นและถอดการค้นหาในข้อความออกทั้งหมด แล้ว 2.17.0 สามรุ่นในไม่กี่สัปดาห์คือภาพของคนที่กำลังไล่แก้ของที่ไม่รู้ว่าตัวเองมีกี่ทาง

ระยะเวลา: ปลั๊กอินค้นหา JNDI ออกมากับ Log4j 2.0-beta9 วันที่ 18 กรกฎาคม 2013 รุ่นที่ได้รับผลกระทบไล่จาก 2.0-beta9 ถึง 2.14.1 ราวแปดปีในโค้ดที่ปล่อยใช้แล้ว ก่อนมีใครรายงาน

ความกว้างวัดได้ และเหตุผลที่มันกว้างขนาดนั้นคือส่วนที่ทำให้แก้ยาก Google วิเคราะห์วันที่ 17 ธันวาคม 2021 พบแพ็กเกจ Java ที่ได้รับผลกระทบเกิน 35,000 แพ็กเกจ เกิน 8% ของ Maven Central ดัชนีแพ็กเกจ Java หลัก ส่วนใหญ่ได้รับผล

กระทบโดยอ้อม ไลบรารีมาถึงลึกกว่าหนึ่งชั้นของ dependency ทีมที่อ่านรายการ dependency ของตัวเองจึงไม่เห็น Log4j อยู่ในนั้นเลย ทั้งที่มันอยู่ Cyber Safety Review Board ในรายงานฉบับแรกวันที่ 11 กรกฎาคม 2022 เรียกมันว่าช่องโหว่ระดับ endemic คาดว่าจะถูกใช้ต่อไปถึงทศวรรษ 2030 ด้วยเหตุผลตรงตัวว่าไม่มีวิธีกลางที่จะหาสำเนาทุกชุด และจึงไม่มีวิธีกลางที่จะแก้

แล้วใครเป็นคนแก้ ผู้ดูแล Apache Logging Services โครงการเบื้องหลังไลบรารีที่ฝังอยู่ในสัดส่วนที่วัดได้ของซอฟต์แวร์ Java ทั้งโลก เป็นอาสาสมัครไม่ได้รับเงิน รายงานร่วมสมัยให้จำนวน 16 คน ตัวเลขนี้ไม่ได้ตรวจกับแหล่งต้นทางของ Apache ส่วนตัวเลขสามคนที่เล่าต่อกันมา หาแหล่งไม่พบ และมันไม่สำคัญเท่าที่คนเล่าคิด ครึ่งที่รับน้ำหนักคือคำว่าไม่ได้รับเงิน ไม่ใช่จำนวนหัว การแก้ไขต้องถูกผลิตโดยอาสาสมัครเหล่านั้น ด้วยความเร็ว ในสุดสัปดาห์เดียว ไม่มีค่าตอบแทน ขณะที่ผู้ขายความมั่นคงทุกรายบนโลกเขียนถึงพวกเขา

ในแง่สันของเล่ม ช่องโหว่หนึ่งช่องคัดลอกไปพร้อมไลบรารีสามหมื่นห้าพันครั้ง โดยที่คนถือสำเนาไม่รู้ว่าถืออยู่ ผู้โจมตีไม่ต้องหาว่าใครใช้ Log4j เขายิงทุกอย่างที่รับข้อความ

สิ่งที่กรณีนี้สอน: สร้างรายการ dependency ที่ลึกกว่าหนึ่งชั้น แล้วค้นชื่อไลบรารีที่คุณไม่เคยตัดสินใจใช้ — ที่นั่นคือที่ช่องโหว่ระดับ endemic อาศัยอยู่

Colonial Pipeline · พฤษภาคม 2021

วันที่ 7 พฤษภาคม 2021 Colonial Pipeline ปิดท่อส่งน้ำมันยาวกว่า 5,500 ไมล์ที่ส่งน้ำมันราว 2.5 ล้านบาร์เรลต่อวัน — ราว 45% ของเชื้อเพลิงฝั่งตะวันออกของสหรัฐ — ด้วยมือของบริษัทเอง ระบบที่ผู้โจมตีทำให้ล่มไม่ใช่ท่อ มันคือระบบออกบิล

ส่วนที่เล่มนี้ต้องการอยู่ตรงนั้น ระบบปฏิบัติการของท่อไม่ได้รับการยืนยันว่าถูกเจาะ Colonial ปิดท่อเองเพื่อความปลอดภัย เพราะระบบออกบิลล่ม และบริษัทบันทึกไม่ได้ว่ากำลังส่งอะไรให้ใคร ความล้มเหลวทางกายภาพจึงเป็นผลของระบบธุรกิจล้ม บวกกับไม่มีวิธีเดินเครื่องแบบลดระดับ ส่วนเผื่อที่ขาดไม่ใช่มาตรการความมั่นคง มันคือแผนสำรองด้วยมือ

ลำดับเหตุการณ์กินเวลาไม่ถึงสัปดาห์ การขโมยข้อมูลเดินอยู่แล้วตั้งแต่วันที่ 6 พฤษภาคม ransomware ถูกปล่อยและท่อถูกปิดวันที่ 7 จ่ายค่าไถ่วันที่ 8 กลับมาส่งเต็มระบบวันที่ 12 ราวห้าวัน ห้าวันนั้นพอให้คิวเติมน้ำมันยาวจากเวอร์จิเนียถึงจอร์เจีย และสายการบินเปลี่ยนเส้นทางไปเติมเชื้อเพลิงที่อื่น

ทางเข้าคือรหัสผ่านของบัญชี VPN เก่าที่เลิกใช้แล้วและไม่มี multi-factor authentication ปัจจัยเดียว บัญชีเดียว ไม่มีใครมีหน้าที่ปลดมัน บัญชีนั้นมีอยู่นานเท่าไรก่อนถูกใช้ ไม่ทราบ และนั่นคือคำตอบของคำถามประจำภาคในกรณีนี้ องค์กรตอบไม่ได้เพราะไม่มีใครนับ ต้นเหตุในหนึ่งประโยค: บัญชีเข้าถึงระยะไกลที่หลับอยู่ มีปัจจัยยืนยันเดียวที่ใช้ซ้ำได้ ไม่เคยถูกปลด และไม่มีอะไรในองค์กรรับผิดชอบว่าต้องรู้ว่ามันยังอยู่

ค่าไถ่ราว 75 BTC ราว 4.4 ล้านดอลลาร์ จ่ายให้กลุ่ม DarkSide วันที่ 8 พฤษภาคม วันที่ 7 มิถุนายน กระทรวงยุติธรรมประกาศยึดได้ 63.7 BTC มูลค่าราว 2.3 ล้านดอลลาร์ 85% ของเหรียญ แต่ราวครึ่งเดียวของเงิน เพราะราคา bitcoin ตกระหว่างเดือนที่จ่ายกับเดือนที่ยึด ประโยคว่าค่าไถ่ถูกกู้คืนจึงจริงสำหรับเหรียญและเท็จสำหรับเงิน และหน่วยที่คุณเลือกนับคือสิ่งที่ตัดสินว่าเรื่องนี้จบดีหรือไม่

ในแง่สันของเล่ม ผู้โจมตีไม่ได้แตะท่อ เขาแตะบัญชีที่ไม่มีใครจำได้ว่ามี และท่อหยุดเพราะไม่มีใครเคยซ้อมเดินมันโดยไม่มีคอมพิวเตอร์

สิ่งที่กรณีนี้สอน: ถามว่าถ้าระบบออกบิลล่มพรุ่งนี้ เครื่องจักรจริงเดินต่อด้วยกระดาษได้กี่วัน — คำตอบนั้นคือส่วนเผื่อจริงของท่อ ไม่ใช่ firewall

XZ Utils · มีนาคม 2024

xz เป็น dependency ของ Linux แทบทุกดิสทริบิวชัน และคนที่ดูแลมันมีอยู่คนเดียว ชื่อ Lasse Collin ไม่ได้รับเงิน ตรวจแพตช์ไม่ทัน และเหนื่อย ปลายปี 2021 มีคนเริ่มสนใจข้อเท็จจริงข้อนั้น ตัวตนชื่อ Jia Tan เริ่มส่งแพตช์เล็กๆ ให้ xz-devel

ปฏิบัติการทั้งหมดใช้เวลาสองปีครึ่ง และเกือบทั้งหมดของมันเป็นเรื่องของคน ไม่ใช่โค้ด เมษายน 2022 ตัวตนที่สองชื่อ Jigar Kumar ปรากฏขึ้นมากดดันให้รับแพตช์ของ

Jia Tan มิถุนายน 2022 Jigar Kumar โพสต์สาธารณะกดดัน Collin ผู้ดูแลเพียงคนเดียว เรื่องตรวจช้าและเรียกร้องให้เพิ่มผู้ดูแล Collin ตอบว่าทรัพยากรของเขามีจำกัดเกินไป และ Jia Tan อาจรับบทบาทใหญ่ขึ้น วันที่ 10 มิถุนายน 2022 สามวันหลังอีเมลกดดัน Collin รวม commit แรกที่ Jia Tan เป็นผู้เขียน Jigar Kumar ไม่ปรากฏตัวอีกเลย งานของเขาเสร็จแล้ว กุมภาพันธ์ 2024 backdoor ออกไปกับรุ่น 5.6.0 วันที่แน่นอนของรุ่นนี้ การค้นคว้าครั้งนี้ยืนยันไม่ได้

พื้นผิวที่ถูกโจมตีไม่ใช่โค้ด มันคือผู้ดูแลที่หมดแรงหนึ่งคน ผลิตคำร้องเรียน ผลิตอาสาสมัคร แล้วปล่อยให้ความเหนื่อยทำที่เหลือ ส่วนตัว backdoor เองก็ไม่ได้อยู่ในที่ที่คนตรวจมอง มันไม่ได้อยู่ในคลัง git มันอยู่ใน tarball ของรุ่นที่ปล่อย ซ่อนในไฟล์ทดสอบและถูกประกอบขึ้นโดยสคริปต์ build ใครอ่าน source สาธารณะเห็นแต่ความว่างเปล่า รูปร่างเดียวกับ SUNSPOT ของ SolarWinds กับ compiler ของ Thompson และกับมาตรา 3.2.4 ของรายงานปี 1974 ห้าสิบปี บทเรียนเดียว ยังไม่ได้เรียน

วันที่ 28 มีนาคม 2024 Andres Freund นักพัฒนา PostgreSQL ที่ Microsoft รายงานต่อรายชื่ออีเมล oss-security ว่า xz/liblzma รุ่น 5.6.0 และ 5.6.1 มี backdoor และสิ่งที่พาเขาไปเจอคือ SSH login ที่ใช้เวลาราว 500 มิลลิวินาทีแทนที่จะเป็นราว 100 เขากำลังวัดประสิทธิภาพบน Debian sid เขาไม่ได้ทำงานความมั่นคง เขารำคาญตัวเลขที่ถอยหลัง ไล่ดู เห็น Valgrind ชี้ข้อผิดพลาดเข้าไปใน liblzma และตามไปถึง tarball ของ xz ปฏิบัติการระดับรัฐที่ผ่านการตรวจโค้ด ผ่านการตรวจรับของดิสทริบิวชัน และผ่านสแกนเนอร์อัตโนมัติทุกตัว ถูกจับด้วยความหน่วง 400 มิลลิวินาทีที่คนหนึ่งคนสังเกตขณะวัดอย่างอื่น

CVSS 10.0 ขนาดความเสียหายแทบเป็นศูนย์ และเป็นศูนย์เพราะวิธีที่มันถูกจับได้เท่านั้น มันไปถึง Fedora Rawhide และ Fedora 41, Debian สาย testing/unstable, openSUSE Tumbleweed และ Kali สำหรับเครื่องที่อัปเดตระหว่าง 26–29 มีนาคม 2024 มันไม่ถึง Debian stable, RHEL, Ubuntu รุ่นออกจริง หรือ SUSE Linux Enterprise ถ้ามันไปถึงรุ่น stable รัศมีคือเซิร์ฟเวอร์ Linux ส่วนใหญ่บนโลก พร้อมช่องรันโค้ดจากระยะไกลก่อนยืนยันตัวตน สำหรับใครก็ตามที่ถือกุญแจส่วนตัวที่ตรงกัน ราวห้าสัปดาห์ในโลกจริง เทียบกับสองปีครึ่งของการเตรียม ส่วนที่แพงของการโจมตี supply chain คือความไว้ใจ ไม่ใช่โค้ด

สิ่งที่กรณีนี้สอน: เมื่อตัวเลขประสิทธิภาพเปลี่ยนโดยไม่มีคำอธิบาย ให้ไล่จนเจอสาเหตุ ไม่ใช่จนมันหายไป — ส่วนเผื่อชิ้นสุดท้ายของระบบนิเวศคือคนที่ทนตัวเลขแปลกไม่ได้

ท่าที่เอาไปใช้ได้: สำหรับทุกระบบที่คุณดูแล เขียนวันที่สองวันไว้ข้างกัน — วันที่ช่องโหว่ประเภทนั้นถูกตีพิมพ์ครั้งแรก กับวันที่ระบบของคุณปิดมัน ช่องว่างระหว่างสองวันคือเวลาที่ผู้โจมตีมีมากกว่าคุณ และในภาคนี้มันวัดเป็นปี

จะพลิกคำตัดสินของภาคนี้ได้เมื่อ: มีชุดข้อมูลที่ตรวจสอบได้ว่าช่องโหว่ในซอฟต์แวร์ที่ใช้กันกว้าง ส่วนใหญ่ถูกใช้ภายในไม่กี่วันหลังเกิด ไม่ใช่หลายปี หรือมีวิธีคำนวณและตัวเลขที่ audit ได้ปรากฏสำหรับหนึ่งหมื่นล้านดอลลาร์ของ NotPetya, 1.2 Tbps ของ Dyn หรือ 100 ล้านดอลลาร์ของหนอน Morris

แหล่งของภาคนี้:

- Karger และ Schell, Multics Security Evaluation: Vulnerability Analysis, ESD-TR-74-193 Vol. II, มิถุนายน 1974 — https://csrc.nist.gov/files/pubs/conference/1998/10/08/proceedings-of-the-21st-nissc-1998/final/docs/early-cs-papers/karg74.pdf
- Karger และ Schell, Thirty Years Later: Lessons from the Multics Security Evaluation, ACSAC 2002 — https://www.acsac.org/2002/papers/classic-multics.pdf
- Thompson, Reflections on Trusting Trust, CACM 27(8), สิงหาคม 1984 — https://www.cs.cmu.edu/~rdriley/487/papers/Thompson_1984_ReflectionsonTrustingTrust.pdf
- Wheeler, Fully Countering Trusting Trust through Diverse Double-Compiling — https://arxiv.org/pdf/1004.5534
- Cox, Running the "Reflections on Trusting Trust" Compiler, 2023 — https://research.swtch.com/nih
- FBI, The Morris Worm: 30 Years Since First Major Attack on the Internet — https://www.fbi.gov/news/stories/morris-worm-30-years-since-first-major-attack-on-internet-110218
- United States v. Morris, 928 F.2d 504 (2d Cir. 1991) — https://law.justia.com/cases/federal/appellate-courts/F2/928/504/452673/
- Washington Post, No Jail Time Imposed in Hacker Case, 5 พฤษภาคม 1990 — https://www.washingtonpost.com/archive/politics/1990/05/05/no-jail-time-imposed-in-hacker-case/3f8fbd6c-e9e3-4249-8c51-bf72c43e2055/
- Aleph One, Smashing the Stack for Fun and Profit, Phrack 49, 8 พฤศจิกายน 1996 — https://phrack.org/issues/49/14

- rain.forest.puppy, NT Web Technology Vulnerabilities, Phrack 54, 25 ธันวาคม 1998 — https://phrack.org/issues/54/8
- กระทรวงยุติธรรมสหรัฐ, คำฟ้อง Albert Gonzalez, สิงหาคม 2009 — https://www.justice.gov/archives/opa/pr/alleged-international-hacker-indicted-massive-attack-us-retail-and-banking-networks
- Heartland Payment Systems, รายงาน 10-Q ไตรมาสสิ้นสุด 31 มีนาคม 2011 — https://www.sec.gov/Archives/edgar/data/0001144354/000114435411000011/hpy0331201110q.htm
- ISIS, Stuxnet Malware and Natanz, 22 ธันวาคม 2010 — https://isis-online.org/isis-reports/stuxnet-malware-and-natanz-update-of-isis-december-22-2010-reportsupa-href1
- CISA, ICS Advisory ICSA-10-238-01B — https://www.cisa.gov/news-events/ics-advisories/icsa-10-238-01b
- Kaspersky, The Man Who Found Stuxnet: Sergey Ulasen, 2 พฤศจิกายน 2011 — https://eugene.kaspersky.com/2011/11/02/the-man-who-found-stuxnet-sergey-ulasen-in-the-spotlight/
- Netcraft, Half a Million Widely Trusted Websites Vulnerable to Heartbleed Bug, เมษายน 2014 — https://www.netcraft.com/blog/half-a-million-widely-trusted-websites-vulnerable-to-heartbleed-bug
- CISA, OpenSSL Heartbleed Vulnerability CVE-2014-0160, 8 เมษายน 2014 — https://www.cisa.gov/news-events/alerts/2014/04/08/openssl-heartbleed-vulnerability-cve-2014-0160
- The Register, Linux Foundation Core Infrastructure Initiative, 24 เมษายน 2014 — https://www.theregister.com/2014/04/24/linux_foundation_core_infrastructure
- The Register, Robin Seggelmann และ Heartbleed, 11 เมษายน 2014 — https://www.theregister.com/2014/04/11/openssl_heartbleed_robin_seggelmann/
- Antonakakis และคณะ, Understanding the Mirai Botnet, USENIX Security 2017 — https://www.usenix.org/system/files/conference/usenixsecurity17/sec17-antonakakis.pdf
- CISA, Heightened DDoS Threat Posed by Mirai and Other Botnets, 14 ตุลาคม 2016 — https://www.cisa.gov/news-events/alerts/2016/10/14/heightened-ddos-threat-posed-mirai-and-other-botnets
- การโจมตี Dyn รวมถึงคำแถลงของ Dyn เรื่อง 1.2 Tbps — https://en.wikipedia.org/wiki/DDoS_attacks_on_Dyn
- GAO-18-559, Data Protection: Actions Taken by Equifax and Federal Agencies, สิงหาคม 2018 — https://www.gao.gov/assets/gao-18-559.pdf
- Equifax, Cybersecurity Incident, 7 กันยายน 2017 — https://investor.equifax.com/news-events/press-releases/detail/237/equifax-releases-details-on-cybersecurity-incident
- Apache Software Foundation, แถลงเรื่อง Struts และ Equifax — https://news.apache.org/foundation/entry/media-alert-the-apache-software

- House Oversight Committee, The Equifax Data Breach, ธันวาคม 2018 — https://oversight.house.gov/wp-content/uploads/2018/12/Equifax-Report.pdf
- CISA, Petya Ransomware, 1 กรกฎาคม 2017 — https://www.cisa.gov/news-events/alerts/2017/07/01/petya-ransomware
- Axios, White House confirms NotPetya attribution, 15 กุมภาพันธ์ 2018 — https://www.axios.com/2018/02/15/white-house-confirms-notpetya-1518728781
- The Register, FedEx NotPetya damages, 20 กันยายน 2017 — https://www.theregister.com/2017/09/20/fedex_notpetya_damages/
- CyberScoop, NotPetya cost Merck — https://cyberscoop.com/notpetya-ransomware-cost-merck-310-million/
- Control Engineering, How NotPetya took down Maersk — https://www.controleng.com/throwback-attack-how-notpetya-accidentally-took-down-global-shipping-giant-maersk/
- CrowdStrike, SUNSPOT Malware Technical Analysis — https://www.crowdstrike.com/en-us/blog/sunspot-malware-technical-analysis/
- SolarWinds Security Advisory FAQ — https://www.solarwinds.com/sa-overview/securityadvisory/faq
- The Record, SolarWinds says fewer than 100 customers were impacted — https://therecord.media/solarwinds-says-fewer-than-100-customers-were-impacted-by-supply-chain-attack
- CyberScoop, Neuberger on SolarWinds, 17 กุมภาพันธ์ 2021 — https://cyberscoop.com/solarwinds-cyber-espionage-russia-neuberger/
- Cyber Safety Review Board, Review of the December 2021 Log4j Event, 11 กรกฎาคม 2022 — https://www.cisa.gov/sites/default/files/publications/CSRB-Report-on-Log4-July-11-2022_508.pdf
- CERT/CC, VU#930724 Log4j — https://www.kb.cert.org/vuls/id/930724
- InfoWorld, How developers scrambled to secure the Log4j vulnerability — https://www.infoworld.com/article/2271392/how-developers-scrambled-to-secure-the-log4j-vulnerability.html
- CISA, Alert AA21-356A — https://www.cisa.gov/news-events/cybersecurity-advisories/aa21-356a
- CISA, The Attack on Colonial Pipeline: What We've Learned — https://www.cisa.gov/news-events/news/attack-colonial-pipeline-what-weve-learned-what-weve-done-over-past-two-years
- กระทรวงพลังงานสหรัฐ, Colonial Pipeline Cyber Incident — https://www.energy.gov/ceser/colonial-pipeline-cyber-incident
- Morrison Foerster, DOJ Retrieves Millions in Colonial Pipeline Ransom, 9 มิถุนายน 2021 — https://www.mofo.com/resources/insights/210609-doj-retrieves-millions
- Cox, Timeline of the xz open source attack — https://research.swtch.com/xz-timeline

- Boehs, Everything I know about the xz backdoor — https://boehs.org/node/everything-i-know-about-the-xz-backdoor
- Help Net Security, xz backdoored: affected distros, 31 มีนาคม 2024 — https://www.helpnetsecurity.com/2024/03/31/xz-backdoored-linux-affected-distros/
- Red Hat warns of backdoor in xz tools — https://www.bleepingcomputer.com/news/security/red-hat-warns-of-backdoor-in-xz-tools-used-by-most-linux-distros/

5 · การป้องกันที่มีหลักฐาน

ภาคนี้ถามคำถามเดียวกับการป้องกันทุกแบบ วัดแล้วได้เท่าไร คำถามสั้นแค่นี้ตอบยากกว่าที่ควรจะเป็น เพราะการป้องกันจำนวนมากที่องค์กรจ่ายเงินซื้ออยู่ทุกปี ไม่เคยมีใครเอาตัวเลขก่อนกับตัวเลขหลังมาวางไว้ข้างกัน มีแต่คำชมและอัตราที่ผู้ขายพิมพ์ให้ดูเอง การป้องกันที่มีตัวเลขก่อนและหลังตีพิมพ์ไว้ จึงอยู่เหนือการป้องกันที่ทุกคนชม เหตุผลอยู่ที่สันของเล่ม — ธรรมชาติไม่ปรับตัว ศัตรูปรับตัว การป้องกันที่ไม่เคยถูกวัดกับผู้โจมตีจริง อาจกำลังกันสิ่งที่ผู้โจมตีเลิกทำไปแล้ว และไม่มีทางรู้จนกว่าจะมีคนนับ

กฎรหัสผ่านที่ผิดมาสามสิบปี · 2017–2025

กฎรหัสผ่านที่คนทำงานทั้งรุ่นท่องได้ขึ้นใจมีอยู่สองข้อ ต้องผสมตัวพิมพ์ใหญ่ ตัวเลข และสัญลักษณ์ และต้องเปลี่ยนใหม่ตามรอบเวลา ทั้งสองข้อไม่ได้มาจากการนับรหัสผ่านที่ถูกเจาะจริง มันมาจากแบบจำลองความสุ่มที่คำนวณบนกระดาษ แล้วมันอยู่ในนโยบายขององค์กรทั่วโลกมาสามสิบปี

เดือนมิถุนายน 2017 NIST ตีพิมพ์ SP 800-63B และลดกฎบังคับผสมตัวอักษรในรหัสผ่านลงเหลือคำว่า SHOULD NOT — ไม่ใช่ห้าม แต่เป็นคำแนะนำว่าไม่ควรทำ พร้อมกำหนดความยาวขั้นต่ำ 8 ตัวอักษร

ข้อความจริงในมาตรา 5.1.1.2 ระบุว่าผู้ตรวจสอบ SHOULD NOT บังคับกฎผสมตัวอักษร และ SHOULD NOT บังคับเปลี่ยนรหัสผ่านตามรอบเวลา แต่ SHALL บังคับเปลี่ยนเมื่อมีหลักฐานว่ารหัสผ่านรั่ว สิ่งที่ใส่เข้ามาแทนบอกทิศทางได้ชัดกว่าสิ่งที่ถอดออก คือความยาว blocklist ของรหัสผ่านที่เคยรั่วแล้ว จำกัดจำนวนครั้งที่ลองผิด และต้องยอมให้วาง (paste) ได้เพื่อให้ password manager ทำงาน ห้าม truncate รหัสผ่าน และควรรับได้ถึง 64 ตัวอักษร ของเก่าคุมสิ่งที่ผู้ใช้พิมพ์ ของใหม่คุมสิ่งที่ผู้โจมตีลองได้

เหตุผลที่ NIST ให้อยู่ในภาคผนวก A และเป็นเหตุผลทางพฤติกรรม ไม่ใช่คณิตศาสตร์ ผู้ใช้ตอบสนองต่อกฎผสมตัวอักษรในแบบที่คาดเดาได้ ความสุ่ม (entropy) ที่กฎ

ดูเหมือนจะซื้อมา จึงไม่ใช่ความสุ่มที่ผู้โจมตีเจอจริง กฎเดิมเผื่อความปลอดภัยไว้บนแกนที่ผู้โจมตีไม่ได้เดิน เพราะผู้ใช้ปรับตัวเข้าหากฎ และผู้โจมตีปรับตัวเข้าหาผู้ใช้ NIST จึงย้ายไปใช้แบบจำลองที่วัดจากรหัสผ่านที่รั่วแล้ว ประโยคที่มักถูกยกมาอ้างจากภาคผนวกนั้นตรวจกับต้นฉบับโดยตรงไม่ได้ในการค้นคว้าครั้งนี้ เล่มนี้จึงอ้างเฉพาะมาตรา 5.1.1 ที่อ่านจากต้นฉบับ

สองความเชื่อที่แพร่อยู่ตายในการค้นคว้า ข้อแรก "NIST ห้ามกฎรหัสผ่านซับซ้อนตั้งแต่ 2017" — ไม่จริง ปี 2017 เป็นแค่ SHOULD NOT คำว่า SHALL NOT มาถึงในฉบับแก้ไขครั้งที่ 4 ที่ประกาศใช้จริงวันที่ 31 กรกฎาคม 2025 ข้อสอง "ฉบับแก้ไขครั้งที่ 4 ออกในปี 2024" — ไม่จริงเช่นกัน ร่างสาธารณะฉบับแรกออก 16 ธันวาคม 2022 ร่างที่สอง 21 สิงหาคม 2024 ฉบับจริง 31 กรกฎาคม 2025 ผู้ขายซอฟต์แวร์ที่อ้างว่าทำตามมาตรฐานปี 2024 กำลังอ้างร่าง

ฉบับที่ 4 ยกคำแนะนำเดิมขึ้นเป็นข้อบังคับ ห้ามกฎผสมตัวอักษร ห้ามบังคับเปลี่ยนตามรอบ และยกความยาวขั้นต่ำเป็น 15 ตัวอักษรสำหรับรหัสผ่านที่ใช้เป็นปัจจัยเดียว 8 ตัวใช้ได้เฉพาะเมื่อรหัสผ่านเป็นหนึ่งในหลายปัจจัย พร้อมข้อสังเกตใหม่ว่า blocklist ที่ใหญ่เกินไปได้ประโยชน์เพิ่มน้อย เพราะมันกันการโจมตีแบบ online ที่การจำกัดจำนวนครั้งกันอยู่แล้ว

รูปร่างของเรื่องนี้สำคัญกว่าตัวกฎ สองในสามกฎที่โด่งดังของปี 2017 ไม่ได้มีหลักฐานใหม่ในปี 2025 มันคือหลักฐานชุดเดิมที่ถูกเลื่อนจากคำแนะนำเป็นข้อบังคับหลังใช้งานจริงแปดปี แปดปีนั้นไม่ได้ใช้หาหลักฐาน แต่ใช้รอให้หลักฐานชุดเดิมหนักพอจะเขียนเป็นคำสั่ง

สิ่งที่กรณีนี้สอน: ก่อนบังคับกฎกับผู้ใช้ ถามว่ากฎนั้นถูกวัดกับรหัสผ่านที่รั่วจริง หรือกับแบบจำลองความสุ่มบนกระดาษ

กุญแจฮาร์ดแวร์กับ MFA

ตัวเลขที่ใช้ขาย MFA และกุญแจฮาร์ดแวร์มาเกือบสิบปีมีอยู่สามตัว เรียงกันในสไลด์เดียวจนฟังดูเหมือนงานวิจัยสามชิ้น พอไล่กลับไปหาต้นทาง ตัวแรกเป็นคำให้สัมภาษณ์ ตัวที่สองเป็นสถิติภายในที่ไม่ได้ตีพิมพ์ ตัวที่สามไม่มีอยู่ในเอกสารของ

หน่วยงานที่ถูกอ้างชื่อ ไม่ได้แปลว่า MFA ไม่ได้ผล — ตัวเลขที่วัดจริงออกมาสูง แต่ช้ากว่าคำขวัญสี่ปี

วันที่ 23 กรกฎาคม 2018 โฆษกของ Google บอกนักข่าว Brian Krebs ว่าพนักงานกว่า 85,000 คนไม่มีใครถูก phishing ยึดบัญชีได้เลย นับตั้งแต่บริษัทบังคับใช้กุญแจฮาร์ดแวร์ต้นปี 2017 และมันคือคำให้สัมภาษณ์ ไม่ใช่งานวิจัย แหล่งหลักอีกแหล่งเป็นประโยคเดียวในบล็อกสินค้าของ Google Cloud วันที่ 30 สิงหาคม 2018 ตอนเปิดตัวกุญแจ Titan: ไม่มีรายงานหรือการยืนยันว่าบัญชีถูกยึด "จาก password phishing" นับตั้งแต่บังคับใช้กุญแจกับพนักงาน ไม่มีวิธีวัด ไม่มีตัวหาร ไม่มีผู้ตรวจสอบอิสระ และขอบเขตถูกจำกัดไว้ที่ password phishing เท่านั้น งานวิจัยจริงของ Google มีฉบับเดียว คือ Lang และคณะในการประชุม Financial Cryptography and Data Security ปี 2016 — มักถูกอ้างผิดว่าเป็น USENIX — รายงานการใช้งานสองปีภายในบริษัท เรื่องเวลาที่ใช้ยืนยันตัวและภาระฝ่ายสนับสนุน ไม่ได้อ้างว่า phishing เป็นศูนย์ ตัวเลขภายในของงานนั้นอ่านได้จากบทคัดย่อเท่านั้นในการค้นคว้าครั้งนี้ เล่มนี้จึงไม่พิมพ์

ตัวเลขที่สองคือ 99.9% ของ Microsoft ที่มาคือบล็อกของ Alex Weinert วันที่ 20 สิงหาคม 2019: บัญชีที่เปิด MFA มีโอกาสถูกยึด "น้อยลงกว่า 99.9%" คำนวณจากข้อมูลภายในที่เห็นความพยายามล็อกอินปลอมราว 300 ล้านครั้งต่อวัน ในปี 2019 และไม่เคยตีพิมพ์ สี่ปีต่อมาจึงมีคนวัดจริง Meyer และคณะจาก Microsoft Research (arXiv 2305.00945, 1 พฤษภาคม 2023) ตรวจบัญชี Azure AD ที่มีพฤติกรรมน่าสงสัย ด้วยวิธี benchmark-multiplier ร่วมกับตรวจบัญชีด้วยมือ ได้ผลว่า MFA ลดความเสี่ยงถูกยึดบัญชี 99.22% ทั้งประชากร และ 98.56% ในบัญชีที่รหัสผ่านรั่วไปแล้ว บัญชีที่เปิด MFA มากกว่า 99.99% ไม่ถูกยึดตลอดช่วงศึกษา แอปยืนยันตัวดีกว่า SMS และทั้งสองดีกว่าไม่มี MFA คำขวัญได้ออกวิ่งไปก่อนสี่ปี โดยไม่ต้องรอให้ใครอนุมัติวิธีวัดให้มัน

ตัวเลขที่สามตายในการค้นคว้า "CISA บอกว่า MFA กัน 99% ของการโจมตี" เอกสาร MFA ของ CISA ที่อ่านในการค้นคว้าครั้งนี้ — แผ่นข้อมูลเดือนมกราคม 2022 และแนวทางวันที่ 31 ตุลาคม 2022 — ไม่มีตัวเลข 99% อยู่เลย ตัวเลขนั้นเป็นของ Microsoft ที่ถูกย้ายชื่อเจ้าของ การย้ายชื่อทำให้สถิติภายในของบริษัทหนึ่ง กลายเป็น

คำแนะนำของหน่วยงานรัฐโดยไม่มีใครวัดอะไรเพิ่ม สิ่งที่ CISA พูดจริงคือให้ใช้ MFA แบบทน phishing ได้ (FIDO/WebAuthn, PKI) และเตือนว่า MFA แบบกดยืนยันบนมือถือกับแบบ SMS ถูกหลอกผ่านได้

คำเตือนข้อหลังนั้นคือสันของเล่มในรูปย่อ ผู้โจมตีปรับตัวเข้าหา SMS และปุ่มกดยืนยันไปแล้ว กุญแจฮาร์ดแวร์คือแบบที่ออกแบบไว้รับการปรับตัวนั้น ส่วนตัวเลขของ Google ยังไม่มีงานวิจัยรองรับจนถึงวันนี้ — ของที่ใช้ได้จริง กับของที่พิสูจน์แล้ว ไม่ใช่ของสิ่งเดียวกัน

สิ่งที่กรณีนี้สอน: เมื่อได้ยินตัวเลขผลของการป้องกัน ให้ถามหาเอกสารที่มีตัวหารและวิธีวัด — คำพูดโฆษกไม่มีทั้งสองอย่าง

HTTPS ทั้งเว็บ · 2015–

ปี 2015 ใครก็ตามที่นั่งอยู่บนเส้นทางระหว่างเบราว์เซอร์กับเซิร์ฟเวอร์ ยังอ่านหน้าเว็บส่วนใหญ่ที่วิ่งผ่านหน้าเขาได้ตามสบาย หน้าเว็บที่ Firefox โหลดผ่าน HTTPS ในช่วงนั้นยังต่ำกว่า 30% ของทั้งหมด สิ่งที่ทำให้กรณีนี้อยู่ในภาคนี้ไม่ใช่ว่าตัวเลขนั้นขยับไปได้เท่าไร แต่เป็นว่ามีคนวัดเส้นนั้นทุกวันตลอดสิบปี ด้วยเครื่องมือที่ระบุชื่อได้

วันที่ 14 กันยายน 2015 Let's Encrypt ออกใบรับรองที่เบราว์เซอร์เชื่อถือใบแรกให้ helloworld.letsencrypt.org โครงการประกาศตัวก่อนหน้านั้นวันที่ 18 พฤศจิกายน 2014 ผู้สนับสนุนก่อตั้งคือ Mozilla, EFF, Cisco, Akamai และ IdenTrust ลายเซ็นไขว้ของ IdenTrust คือสิ่งที่ทำให้ใบรับรองใบแรกได้รับความเชื่อถือ เพราะใบรับรองที่ไม่มีใครที่เบราว์เซอร์รู้จักอยู่แล้วรับรองให้ ก็เป็นแค่ไฟล์ จากนั้นเปิดเบต้าสาธารณะ 3 ธันวาคม 2015 ใบที่หนึ่งล้านออกเดือนมีนาคม 2016 ใบที่หนึ่งร้อยล้านเดือนมิถุนายน 2017 ถึงหนึ่งล้านใบต่อวันเดือนกันยายน 2018 ใบที่หนึ่งพันล้านเดือนกุมภาพันธ์ 2020 และปลายเดือนกันยายน 2025 ออกได้สิบล้านใบในวันเดียว

แต่จำนวนใบไม่ใช่เป้าหมาย ISRG ผู้ดำเนินโครงการเขียนไว้เองในบทความครบสิบปีวันที่ 9 ธันวาคม 2025 ว่าสิ่งที่วัดคือสัดส่วนหน้าเว็บที่โหลดผ่าน HTTPS จาก Firefox Telemetry ค่าเฉลี่ยเคลื่อนที่ 14 วัน ตัวเลขในคำของ Josh Aas: ใช้เวลาราวห้าปีดันสัดส่วนทั่วโลกจากต่ำกว่า 30% ไปถึงราว 80% แล้วนิ่งอยู่ตรงนั้นตั้งแต่นั้นมา ใน

สหรัฐอยู่ใกล้ 95% มาระยะหนึ่งแล้ว ตัวเลขรายปีพร้อมวันที่ไม่ได้มาจากแหล่งต้นทางในการค้นคว้าครั้งนี้ หน้าสถิติของ Let's Encrypt วาดกราฟด้วย JavaScript และรายงานของ Google หยุดที่ข้อมูลเดือนกรกฎาคม 2022 เล่มนี้จึงพิมพ์เฉพาะตัวเลขที่ ISRG เขียนเอง

ต้องพูดให้ชัดว่าเส้นกราฟเส้นนี้วัดอะไร HTTPS เป็นการป้องกันที่วัดการใช้ ไม่ได้วัดการโจมตีที่ถูกกัน สัดส่วน 80% บอกว่าสายส่งถูกเข้ารหัสไปแค่ไหน มันไม่ได้บอกว่าผู้ดักฟังที่เคยอ่านสายส่งย้ายไปทำอะไรต่อ ข้อดีของมันอยู่ที่อย่างอื่น คือมันเป็นการป้องกันที่มีเส้นกราฟก่อน-หลังยาวสิบปี จากเครื่องวัดที่ระบุชื่อได้ ซึ่งเป็นของที่การป้องกันส่วนใหญ่ในภาคนี้ไม่มี

และของที่หายากกว่าพื้นคือเพดาน 20% สุดท้ายไม่ขยับมาห้าปี ISRG บอกตรงๆ ว่าไม่รู้ว่ามันคืออะไร เดาว่าเป็นเครือข่ายภายในองค์กรและเว็บส่วนตัว แล้วเรียกมันว่าคำถามวิจัยที่ยังเปิดอยู่ องค์กรที่ดันตัวเลขของตัวเองขึ้นไปได้ห้าสิบจุดแล้วเขียนว่าไม่รู้ว่าอะไรขวางอยู่ กำลังทำสิ่งที่คนขายของไม่ทำ การป้องกันที่ผู้ดำเนินการวัดเพดานของตัวเองแล้วพิมพ์ออกมา มีค่ากว่าการป้องกันที่วัดได้แค่พื้น

สิ่งที่กรณีนี้สอน: ถามว่าตัวชี้วัดของการป้องกันวัดการใช้หรือวัดผล แล้วถามต่อว่าเพดานของมันอยู่ตรงไหน และใครรู้ว่าทำไม

ภาษาที่ปลอดภัยกับหน่วยความจำ · 2019–2024

ช่องโหว่ประเภท memory safety ถูกปฏิบัติมาตลอดเหมือนสภาพอากาศ คือเรื่องที่ต้องวางแผนอยู่กับมัน ไม่ใช่เรื่องที่จะทำให้หมดไป ทุกคนรู้ว่ามันคือสัดส่วนใหญ่ที่สุดของบั๊กด้านความปลอดภัย ทุกคนรู้วิธีลดมันทีละตัว และไม่มีใครมีกราฟที่แสดงว่ามันเล็กลงทั้งประเภท

วันที่ 25 กันยายน 2024 Google รายงานจากข้อมูล security bulletin ของตัวเองว่าช่องโหว่ประเภท memory safety เคยเป็น 76% ของช่องโหว่ทั้งหมดใน Android ปี 2019 และในปี 2024 เหลือ 24% ต่ำกว่าค่ากลางของอุตสาหกรรมที่ 70% และยังลดต่อ สิ่งที่หายากในเรื่องนี้ไม่ใช่ตัวเลข แต่เป็นรูปร่างของมัน — บั๊กทั้งประเภทหนึ่งค่อยๆ ตายลงบนกราฟ ในสาขาที่คุ้นกับการเห็นเส้นแบนมาตลอด

ส่วนที่ขัดกับสามัญสำนึกคือ Android ไม่ได้เขียนโค้ด C/C++ เก่าใหม่ ตั้งแต่ราวปี 2019 งานพัฒนาใหม่ย้ายไปภาษาที่ปลอดภัยกับหน่วยความจำ โค้ดเก่าถูกปล่อยไว้แก้เฉพาะตอนมีบั๊ก และจำนวนจริงลดด้วย ไม่ใช่แค่สัดส่วน ปี 2024 ประมาณการไว้ที่ 36 CVE ประเภทนี้ทั้งปี นับได้ 27 ถึง bulletin เดือนกันยายน

กลไกที่ Google เสนออธิบายว่าทำไมการไม่แตะโค้ดเก่าถึงได้ผล ช่องโหว่สลายตัวตามอายุโค้ด โค้ดอายุห้าปีมีความหนาแน่นของช่องโหว่ต่ำกว่าโค้ดใหม่ 3.4 เท่า (จากงานภายนอก) ถึง 7.4 เท่า (จากที่สังเกตใน Android และ Chromium) หลักฐานที่อ้างคือ Alexopoulos และคณะ USENIX Security 2022 เรื่องอายุขัยของช่องโหว่ในโค้ด เมื่อของใหม่ไม่ไหลเข้า สต็อกเก่าก็เสื่อมไปเอง หยุดกระแสเข้าจึงชนะการล้างสต็อก และมันชนะโดยไม่ต้องขอ budget ไปเขียนของที่ใช้งานได้อยู่แล้วใหม่ทั้งกอง ตัวเลขข้างเคียงสองตัว: อัตรา rollback ของการเปลี่ยนแปลงที่เขียนด้วย Rust ต่ำกว่าของ C++ ไม่ถึงครึ่ง และตัวสร้าง QR ของ Chromium ที่เขียนด้วย Rust เร็วขึ้น 20 เท่าเมื่อถอด sandbox ที่ไม่จำเป็นแล้วออก ความปลอดภัยซื้อประสิทธิภาพที่มาตรการรุ่นก่อนเคยกินไปคืนมา

ค่า 70% ของอุตสาหกรรมต้องติดดาวไว้ มันมาจาก Matt Miller แห่ง MSRC ในงาน BlueHat IL เดือนกุมภาพันธ์ 2019: ราว 70% ของ CVE ที่ Microsoft แก้ในแต่ละปีตลอดราวสิบสองปีก่อนหน้า เป็นปัญหา memory safety ตัวเลขนี้กับค่ากลางที่ Google อ้างไม่ใช่สองแหล่งอิสระ มันคือสมอตัวเดียวกัน วันที่ 6 ธันวาคม 2023 CISA, NSA, FBI และหน่วยงานพันธมิตรตีพิมพ์ The Case for Memory Safe Roadmaps ภายใต้โครงการ Secure by Design

ความเชื่อที่ตายในการค้นคว้า: "Android แก้ memory safety ด้วยการเขียน C++ ใหม่เป็น Rust" Google เขียนไว้ตรงข้าม โค้ดเก่าแทบไม่ถูกแตะ ที่ย้ายคือโค้ดใหม่เท่านั้น ในแง่สันของเล่ม นี่คือการป้องกันที่ไม่ได้เพิ่มด่านให้ผู้โจมตีเดินอ้อม แต่เอาชั้นของบั๊กออกจากโค้ดที่เขียนหลังจากนั้น ผู้โจมตีปรับตัวหาช่องโหว่ที่ไม่มีอยู่ไม่ได้

สิ่งที่กรณีนี้สอน: ปิดทางเข้าของบั๊กประเภทหนึ่งในโค้ดใหม่ก่อนคิดเขียนโค้ดเก่าใหม่ แล้ววัดด้วยสัดส่วนของช่องโหว่รายปี ไม่ใช่จำนวนบรรทัดที่ย้าย

หลักแปดข้อของ Saltzer และ Schroeder · 1975

หลักออกแบบความปลอดภัยแปดข้อที่หลักสูตรทุกหลักสูตรสอน และที่องค์กรเขียนไว้ในนโยบายว่ายึดถือ มาจากบทความเดียวเมื่อห้าสิบปีก่อน เดือนกันยายน 1975 J. H. Saltzer และ M. D. Schroeder ตีพิมพ์มันใน Proceedings of the IEEE ฉบับที่ 63 เล่มที่ 9 ด้วยประโยคเปิดว่าไม่มีใครรู้วิธีสร้างระบบที่ไม่มีข้อบกพร่อง ทางเลือกจึงเป็นหลักที่ทำให้ข้อบกพร่องน้อยลงและเบาลง ห้าสิบปีต่อมา คำถามที่ตอบได้คือข้อไหนถูกสร้างจริง และคำตอบมีรูปร่างชัดเจนมาก ข้อที่ถูกใช้คือข้อที่เพิ่มการควบคุม ข้อที่ถูกทิ้งคือข้อที่บังคับให้สละบางอย่าง

แปดข้อในคำของพวกเขาเอง: economy of mechanism — ออกแบบให้เล็กและง่ายที่สุด fail-safe defaults — ตัดสินสิทธิ์จากการอนุญาต ไม่ใช่จากการยกเว้น complete mediation — ทุกการเข้าถึงทุกวัตถุต้องถูกตรวจสิทธิ์ open design — แบบต้องไม่เป็นความลับ separation of privilege — กลไกที่ต้องใช้สองกุญแจแข็งแรงกว่ากุญแจเดียว least privilege — ทุกโปรแกรมและทุกผู้ใช้ทำงานด้วยสิทธิ์น้อยที่สุดที่งานต้องการ least common mechanism — ลดกลไกที่ผู้ใช้หลายคนใช้ร่วมกัน และ psychological acceptability — ส่วนติดต่อผู้ใช้ต้องง่ายพอให้คนใช้การป้องกันถูกต้องโดยอัตโนมัติ

สิ่งที่มักหายไปตอนถูกอ้างคืออีกสองข้อที่ผู้เขียนระบุเองว่าใช้กับคอมพิวเตอร์ได้ไม่เต็มที่ ข้อแรก work factor — เทียบต้นทุนที่ต้องใช้เจาะกลไกกับทรัพยากรของผู้โจมตี ข้อโต้แย้งของพวกเขาเอง: กลไกป้องกันในคอมพิวเตอร์ส่วนใหญ่คำนวณ work factor ตรงๆ ไม่ได้ เพราะมันถูกเจาะด้วยทางอ้อม รอฮาร์ดแวร์เสีย หรือหาข้อผิดพลาดในโค้ด และไม่มีใครประมาณเวลาของทางอ้อมได้ นี่คือประโยคปี 1975 จากคนเขียนหลักการเอง ว่าความมั่นคงปลอดภัยของระบบวัดแบบ cryptanalysis ไม่ได้ — เขียนไว้ก่อนอุตสาหกรรมที่ขายคะแนนความเสี่ยงเป็นตัวเลขเดียวจะเกิดขึ้นหลายสิบปี ข้อสอง compromise recording — บันทึกให้ได้ว่าถูกเจาะ แทนที่จะกัน ข้อโต้แย้ง: ผู้โจมตีที่ฉลาดพอลบบันทึกนั้นได้ด้วย อุตสาหกรรม audit log และ SIEM ยืนอยู่บนหลักที่ผู้เขียนตัดออก

ห้าสิบปีผ่านไป ข้อไหนถูกใช้จริง — ต่อไปนี้เป็นการประเมิน ไม่ใช่การวัด least privilege ถูกใช้โดยเรียกชื่อตรงๆ ใน IAM, sandbox, container และ sudo fail-safe

defaults กลายเป็น firewall แบบปฏิเสธก่อน open design ถูกใช้ในวิทยาการเข้ารหัสสาธารณะ complete mediation ถูกยอมรับในหลักการและถูกละเมิดเพื่อความเร็ว — cache ผลตรวจสิทธิ์และบั๊ก TOCTOU คือรูปที่ผู้เขียนทำนาย separation of privilege ได้ครึ่งเดียว ในรูป MFA และกฎสองคน

สองข้อถูกทิ้ง economy of mechanism — ระบบสมัยใหม่คือขั้วตรงข้ามของเล็กและง่าย และ least common mechanism — cloud ที่ผู้เช่าหลายรายใช้ร่วม kernel ร่วม cache ของ CPU ร่วม Spectre, Meltdown และ Rowhammer คือการละเมิดข้อนี้โดยมีซิลิคอนรองอยู่ข้างใต้ ทั้งสองข้อไม่ได้ถูกทิ้งเพราะไม่รู้ แต่เพราะราคา ข้อหนึ่งขอให้เลิกใส่ฟีเจอร์ อีกข้อขอให้เลิกแบ่งเครื่องกันใช้ และไม่มีใครอยากจ่ายทั้งสองอย่าง

สิ่งที่กรณีนี้สอน: ในรายการหลักการที่องค์กรอ้างว่ายึดถือ ขีดเส้นใต้ข้อที่บังคับให้เลิกทำบางอย่าง — ข้อนั้นคือข้อที่ยังไม่ได้ทำ

การเปิดเผยช่องโหว่แบบประสานงาน

คนที่พบช่องโหว่กับคนที่ต้องแก้มันไม่เคยอยู่ภายใต้เส้นตายเดียวกัน ผู้ขายเป็นคนกำหนดว่าจะแก้เมื่อไร ส่วนคนที่รายงานเข้าไปมีทางเลือกอยู่สองทาง คือรอ หรือเปิดเผยเองแล้วรับผลที่ตามมา สิ่งที่ CERT/CC ทำในปี 2000 คือเอานาฬิกามาวางไว้ตรงกลาง

วันที่ 9 ตุลาคม 2000 นโยบายของ CERT/CC มีผลบังคับ: ช่องโหว่ที่รายงานเข้ามาจะถูกเปิดเผยต่อสาธารณะหลัง 45 วันนับจากรายงานครั้งแรก ไม่ว่าจะมีแพตช์หรือไม่ Shawn Hernan หัวหน้าทีมจัดการช่องโหว่ของ CERT/CC ประกาศบน BugTraq วันที่ 4 ตุลาคม 2000 ข้อยกเว้นมีสามแบบ ถูกโจมตีจริงอยู่แล้ว ภัยร้ายแรงเป็นพิเศษหรือเบาเป็นพิเศษ และช่องโหว่ที่ต้องแก้มาตรฐานที่ใช้อยู่

จุดสำคัญของแบบนี้คือนาฬิกาไม่ขึ้นกับแพตช์ มันไม่ใช่คำสัญญาต่อผู้ขาย มันคือเส้นตายที่บังคับผู้ขาย และนั่นคือสิ่งที่ทำให้มันใช้ประสานงานได้จริง นาฬิกา 45 วันยังเป็นนโยบายที่ CERT/CC พิมพ์อยู่จนวันนี้ และเป็นจุดอ้างอิงที่กรอบเวลาอื่นวัดตัวเองเทียบ — Project Zero ของ Google ใช้ 90 วัน ZDI ใช้ 120 วัน

ขนาดของตลาดที่ตามมาวัดได้ HackerOne จ่ายค่าตอบแทนสะสมเกิน 300 ล้านดอลลาร์ ณ เดือนตุลาคม 2023 ตามแถลงของบริษัทเอง Google จ่ายผ่านโครงการ VRP 11.8 ล้านดอลลาร์ให้ผู้วิจัย 660 คนในปี 2024 ตัวเลขนี้อ่านจากรายงานข่าวที่อ้างบล็อกของ Google ไม่ใช่จากบล็อกโดยตรง

แล้วมันลดเหตุการณ์ได้ไหม คำตอบตรงคือยังไม่มีใครแสดง สิ่งที่วรรณกรรมวัดคืออัตราการค้นพบและต้นทุนต่อบั๊ก ซึ่งเป็นคนละเรื่องกัน Finifter, Akhawe และ Wagner ใน USENIX Security 2013 คำนวณว่าโครงการของ Chrome มีต้นทุนราว 485 ดอลลาร์ต่อวัน ของ Mozilla ราว 658 ดอลลาร์ต่อวัน และโครงการหนึ่งได้รายงานที่ใช้ได้ราว 0.429 ฉบับต่อวัน หรือราว 156 ต่อปี ข้อสรุปของพวกเขาเป็นเรื่องเศรษฐศาสตร์ล้วนๆ คือถูกกว่าจ้างนักวิจัยเต็มเวลาจำนวนเท่ากัน ไม่ใช่ข้อสรุปเรื่องเหตุการณ์ที่เลี่ยงได้ Walshe และ Simpson จาก Oxford ปี 2020 ได้ทิศทางเดียวกัน ต้นทุนรายปีของโครงการต่ำกว่าจ้างวิศวกรเพิ่มสองคน และความสำเร็จของโครงการสัมพันธ์กับค่าตอบแทนเฉลี่ยเพียงอ่อนๆ ส่วนงานศึกษา Chromium และ Firefox ใน ACM Web Conference 2023 บอกว่าทำไมมันถึงแทนกันไม่ได้ บั๊กที่นักล่าภายนอกเจอต่างจากที่ทีมภายในเจอ และต่างจากที่ผู้โจมตีเจอ bug bounty จึงเสริมการตรวจภายใน ไม่ได้แทน

ข้อความที่ป้องกันได้จึงเป็นแบบนี้ bug bounty เป็นช่องทางค้นพบที่ถูกและวัดได้ดี ส่วนข้อความว่ามันลดการเจาะระบบ ไม่มีค่าประมาณเชิงสาเหตุตีพิมพ์ในทางใดทางหนึ่ง มันเป็นคำถามที่ยังเปิด ไม่ใช่คำตอบว่าไม่ ในแง่สันของเล่ม นาฬิกา 45 วันคือการป้องกันที่ออกแบบจากข้อเท็จจริงว่าผู้โจมตีปรับตัว มันแข่งกับเขา ไม่ได้รอผู้ขาย

สิ่งที่กรณีนี้สอน: แยกตัวเลขที่วัดการค้นพบออกจากตัวเลขที่วัดเหตุการณ์ ก่อนใช้ตัวแรกจ่ายเงินซื้อตัวหลัง

fuzzing · 2016–

วิธีหาช่องโหว่ที่ผู้โจมตีใช้ได้เสมอคือยิงอินพุตสุ่มใส่โค้ดจนมันพัง fuzzing คือฝ่ายป้องกันทำสิ่งเดียวกันนั้น แต่ทำก่อน และในกรณีนี้ทำฟรีให้โครงการที่ไม่มีเงินจ้างใคร

มาทำ แรงจูงใจที่ประกาศไว้คือ Heartbleed — บั๊กในโค้ดที่คนอื่นใช้ต่อกันทั้งอินเทอร์เน็ต และไม่มีใครเป็นเจ้าของงบประมาณสำหรับตรวจมัน

วันที่ 1 ธันวาคม 2016 Google เปิดตัว OSS-Fuzz และถึงเดือนพฤษภาคม 2025 README ของโครงการนับช่องโหว่ที่เจอได้เกิน 13,000 รายการ บั๊กรวมเกิน 50,000 รายการ ในโครงการโอเพนซอร์ส 1,000 โครงการ โครงการเดินร่วมกับ Core Infrastructure Initiative และต่อมา OpenSSF เครื่องมือคือ libFuzzer, AFL++ และ Honggfuzz ทำงานคู่กับ sanitizer และ ClusterFuzz ที่กระจายงานไปหลายเครื่อง ภาษาที่รับคือ C/C++, Rust, Go, Python, Java, JavaScript และ Lua

เส้นทางของตัวเลขบอกอัตราการเติบโต เดือนสิงหาคม 2023 บล็อกของ Google รายงานช่องโหว่เกิน 10,000 และบั๊กเกิน 36,000 ในราว 1,000 โครงการ เดือนพฤศจิกายน 2024 fuzz target ที่ AI สร้างเพิ่มโค้ดที่ถูกทดสอบกว่า 370,000 บรรทัดใน 272 โครงการ C/C++ ปัจจุบัน OSS-Fuzz ทดสอบให้เกิน 1,300 โครงการโดยผู้ดูแลไม่ต้องจ่าย ตัวเลข "40,500 บั๊กใน 650 โครงการ" จากเดือนกรกฎาคม 2022 ที่แพร่อยู่ ขัดกับตัวเลข 36,000 ของปีถัดมา วิธีนับเปลี่ยนตรงไหนไม่ทราบ เล่มนี้ใช้ตัวเลขใน README ที่ Google ยืนยันเองในปัจจุบัน

น้ำหนักถ่วงมาจาก Google เอง และนี่คือส่วนที่ทำให้กรณีนี้ต่างจากการป้องกันอื่น บทความเรื่อง memory safety ของ Android วันที่ 25 กันยายน 2024 จัด fuzzing เป็นมาตรการรุ่นที่สาม "ค้นหาช่องโหว่เชิงรุก" และเขียนว่าวิธีแบบนี้แก้อาการของ memory unsafety ไม่ใช่ต้นเหตุ มันต้องกดดันทีมตลอดเวลาให้ fuzz คัดกรอง และแก้สิ่งที่เจอ ผลที่ตามมาคือความครอบคลุมต่ำ และแม้ทำอย่างทั่วถึง fuzzing ก็ไม่ให้ความมั่นใจสูง — หลักฐานคือช่องโหว่ที่ยังเจอในโค้ดที่ถูก fuzz มาแล้วมาก คาดการณ์ของ Google คือจะพึ่ง fuzzing น้อยลงเมื่อโค้ดที่ปลอดภัยกับหน่วยความจำแพร่ออกไป และได้ผลสูงขึ้นบนพื้นผิวที่เล็กลง

ตัวเลข 13,000 จึงต้องอ่านให้ถูก มันวัดสิ่งที่เจอ ไม่ได้วัดสิ่งที่เหลือ และคนที่เตือนเรื่องนี้ดังที่สุดคือผู้ดำเนินการเอง การป้องกันที่เจ้าของพิมพ์เพดานของตัวเองออกมาเป็นของหายากพอที่จะนับได้

สิ่งที่กรณีนี้สอน: อ่านหาเพดานที่ผู้สร้างเครื่องมือประกาศเอง ก่อนอ่านจำนวนบั๊กที่มันจับได้

การป้องกันที่ไม่มีหลักฐานรองรับ

ไม่มีใครซื้อการอบรม security awareness เพราะความงมงาย คนที่เซ็นใบสั่งซื้อมีหลักฐานอยู่ในมือจริงๆ สิ่งที่ผิดจึงไม่ใช่การตัดสินใจ แต่เป็นตัวเลขที่หลักฐานชุดนั้นวัด

ปี 2025 การทดลองแบบสุ่มควบคุมนาน 8 เดือนกับพนักงานกว่า 19,500 คนของ UC San Diego Health พบว่าการอบรม security awareness ประจำปี ไม่มีความสัมพันธ์กับโอกาสที่พนักงานจะตกเป็นเหยื่อ phishing จำลอง Ho และคณะตีพิมพ์ใน IEEE Symposium on Security and Privacy 2025 ส่ง phishing จำลอง 10 รอบ และเป็นหนึ่งในสองงานเท่านั้นที่สุ่มจัดกลุ่มว่าใครได้อบรม ผลข้อแรก คนที่เพิ่งอบรมเสร็จเมื่อวานกับคนที่อบรมมาสิบเอ็ดเดือนก่อน คลิกพอกัน ผลข้อสอง บทเรียนที่เด้งขึ้นทันทีเมื่อคลิกผิด ลดอัตราพลาดได้ 1.7% เทียบกับกลุ่มควบคุม ผลข้อสาม เหยื่อล่อธรรมดาดึงคนได้ 1–2% เหยื่อล่อที่ทำดีดึงได้เกิน 30% ตัวเลข 1.7% อ่านจากบทคัดย่อและแถลงของ UCSD ไม่ใช่จากตัวบทความ

ตัวเลขที่ฆ่าการป้องกันนี้ไม่ใช่ 1.7% แต่เป็นอัตราส่วนของสองผลหลัง ตัวแปรที่ฝ่ายป้องกันคุมได้ — การอบรม — ขยับผลราว 2 จุด ตัวแปรที่ผู้โจมตีคุมได้ — คุณภาพเหยื่อล่อ — ขยับราว 29 จุด ศัตรูถือตัวแปรที่แรงกว่าราวสิบห้าเท่า ต่อให้ฝ่ายป้องกันดันตัวแปรของตัวเองจนสุดทาง อีกฝั่งก็ยังกว้างกว่า ETH Zurich ได้ผลทางเดียวกันก่อนหน้านั้น Lain และคณะใน IEEE S&P 2022 ตามพนักงานกว่า 14,000 คนนาน 15 เดือน พบว่าการอบรมแบบฝังในเหตุการณ์ไม่ได้ผล และในบางเงื่อนไขทำให้แย่ลง — คนที่ได้บทเรียนหลังคลิกพลาด คลิกในรอบถัดไปมากกว่า

แล้วข้อความว่า "การอบรมได้ผล" มาจากไหน จากงานทบทวน 69 การศึกษาใน Computers & Security ปี 2024 ที่รายงานผลรวมแรง แต่งานส่วนใหญ่ในนั้นวัดความรู้และความตั้งใจ ไม่ได้วัดการคลิก และบ่อยครั้งไม่มีกลุ่มควบคุม สองงานใหญ่ที่มีกลุ่ม

ควบคุมและวัดพฤติกรรม ได้ผลใกล้ศูนย์ทั้งคู่ ส่วนตัวเลขลดคลิกได้ถึง 86% ที่ผู้ขายพิมพ์ เป็นตัวเลขก่อน-หลังที่ไม่มีกลุ่มควบคุม จากคนที่ขายการอบรมเอง

ของชิ้นที่สองคือ antivirus แบบลายเซ็น และมันซ้ำรูปเดิม การประเมิน AI ATAC 1 ของ Oak Ridge National Laboratory ที่กองทัพเรือสหรัฐให้ทุน (arXiv 2308.14835, สิงหาคม 2023) ทดสอบเครื่องมือตรวจจับเชิงพาณิชย์กับไฟล์ 100,000 ไฟล์ ครึ่งดีครึ่งร้าย ในนั้นมีไฟล์รันได้ที่ไม่เคยมีใครเห็น (zero-day) ราว 1,000 ไฟล์ เครื่องมือแบบลายเซ็นจับ zero-day ได้ราว 4% แบบ machine learning ได้ราว 40% หนึ่งในยี่สิบห้ากับสองในห้า ไม่มีตัวไหนใกล้โฆษณา Botacin และคณะใน Computers & Security ปี 2020 ส่งมัลแวร์ที่เพิ่งเก็บใหม่ 28,875 ตัวอย่างขึ้น VirusTotal ติดต่อกัน 30 วัน และพบว่าการประเมิน antivirus เกือบทั้งหมดลดเหลือตัวเลขเดียวคืออัตราตรวจจับ ไม่วัดเวลากว่าจะจับได้ ไม่วัดตัวอย่างที่เคยจับได้แล้วกลับหลุด

คนที่ซื้อของสองอย่างนี้มีเหตุผลรองรับ การอบรมมีงานทบทวน 69 ชิ้นหนุน antivirus มีอัตราตรวจจับที่ผู้ขายพิมพ์ให้ดู สิ่งที่ผิดคือตัวเลขที่วัดผิดตัว — วัดความตั้งใจแทนการคลิก วัดมัลแวร์เก่าแทนมัลแวร์ใหม่ ทั้งสองกรณี ตัวที่ไม่ได้วัดคือตัวที่ผู้โจมตีถือ

สิ่งที่กรณีนี้สอน: ก่อนต่ออายุการป้องกันใดๆ ขอตัวเลขก่อน-หลังที่มีกลุ่มควบคุม จากคนที่ไม่ได้ขายมัน แล้วถามว่าตัวเลขนั้นวัดพฤติกรรมหรือวัดความตั้งใจ

ท่าที่เอาไปใช้ได้: เขียนรายการการป้องกันที่องค์กรจ่ายเงินซื้ออยู่ แล้วใส่ตัวเลขก่อน-หลังที่มีกลุ่มควบคุมจากคนที่ไม่ได้ขายมันไว้ข้างแต่ละรายการ ช่องที่ว่างไม่ได้แปลว่าใช้ไม่ได้ มันแปลว่ายังไม่มีใครวัด และงบก้อนถัดไปควรไปที่การวัดก่อนซื้อเพิ่ม

จะพลิกคำตัดสินของภาคนี้ได้เมื่อ: มีการทดลองแบบสุ่มควบคุมขนาดใกล้เคียง 19,500 คน ที่พบว่า training ลดอัตราคลิกได้มากกว่าที่คุณภาพเหยื่อล่อขยับ หรือมีการประเมินอิสระขนาด 100,000 ไฟล์ที่ signature detection จับไฟล์รันได้ที่ไม่เคยเห็นมาก่อนได้เกินครึ่ง หรือ security bulletin ของ Android รายงานสัดส่วน memory safety กลับขึ้นเหนือ 50% โดยวิธีนับไม่เปลี่ยน

แหล่งของภาคนี้:

- NIST SP 800-63B (2017) มาตรา 5.1.1 — https://pages.nist.gov/800-63-3/sp800-63b.html
- NIST SP 800-63B-4 (2025) — https://pages.nist.gov/800-63-4/sp800-63b.html
- ประวัติการตีพิมพ์ SP 800-63B-4: ร่างแรก 16 ธ.ค. 2022, ร่างสอง 21 ส.ค. 2024, ฉบับจริง 31 ก.ค. 2025 — https://csrc.nist.gov/pubs/sp/800/63/b/4/final
- Google Cloud blog เปิดตัว Titan key, 30 สิงหาคม 2018 — https://cloud.google.com/blog/products/identity-security/titan-security-keys-now-available-on-the-google-store
- Krebs on Security, 23 กรกฎาคม 2018 — https://krebsonsecurity.com/2018/07/google-security-keys-neutralized-employee-phishing/
- Lang และคณะ, Security Keys, FC 2016 — https://research.google.com/pubs/archive/45409.pdf
- Weinert, Microsoft Security blog, 20 สิงหาคม 2019 — https://www.microsoft.com/en-us/security/blog/2019/08/20/one-simple-action-you-can-take-to-prevent-99-9-percent-of-account-attacks/
- Meyer และคณะ, How effective is multifactor authentication at deterring cyberattacks?, 1 พฤษภาคม 2023 — https://arxiv.org/abs/2305.00945
- CISA, Multi-Factor Authentication Fact Sheet, มกราคม 2022 — https://www.cisa.gov/sites/default/files/publications/MFA-Fact-Sheet-Jan22-508.pdf
- CISA, Implementing Phishing-Resistant MFA — https://www.cisa.gov/sites/default/files/publications/fact-sheet-implementing-phishing-resistant-mfa-508c.pdf
- Let's Encrypt ครบสิบปี, 9 ธันวาคม 2025 — https://letsencrypt.org/2025/12/09/10-years
- Let's Encrypt สถิติ (Firefox Telemetry) — https://letsencrypt.org/stats/
- Aas และคณะ, Let's Encrypt, ACM CCS 2019 — https://jhalderm.com/pub/papers/letsencrypt-ccs19.pdf
- Google Security Blog, memory safety ใน Android, 25 กันยายน 2024 — https://security.googleblog.com/2024/09/eliminating-memory-safety-vulnerabilities-Android.html
- Alexopoulos และคณะ, How Long Do Vulnerabilities Live in the Code?, USENIX Security 2022 — https://www.usenix.org/conference/usenixsecurity22/presentation/alexopoulos
- MSRC, A proactive approach to more secure code, กรกฎาคม 2019 — https://msrc.microsoft.com/blog/2019/07/a-proactive-approach-to-more-secure-code/
- CISA และพันธมิตร, The Case for Memory Safe Roadmaps, 6 ธันวาคม 2023 — https://www.cisa.gov/sites/default/files/2023-12/The-Case-for-Memory-Safe-Roadmaps-508c.pdf
- Saltzer และ Schroeder, The Protection of Information in Computer Systems, 1975 — https://www.cs.virginia.edu/~evans/cs551/saltzer/
- นโยบายเปิดเผยช่องโหว่ของ CERT/CC — https://certcc.github.io/certcc_disclosure_policy/

- Finifter, Akhawe และ Wagner, USENIX Security 2013 — https://dl.acm.org/doi/10.5555/2534766.2534790
- Walshe และ Simpson, IEEE 2020 — https://ieeexplore.ieee.org/document/9034828/
- กรณีศึกษา Chromium และ Firefox, ACM Web Conference 2023 — https://dl.acm.org/doi/10.1145/3543507.3583352
- HackerOne แถลงยอดสะสม 300 ล้านดอลลาร์, ตุลาคม 2023 — https://www.hackerone.com/press-release/hackers-surpass-300-million-all-time-earnings-hackerone-platform
- Google VRP ปี 2024 ตามรายงานข่าว — https://www.securityweek.com/google-paid-out-12-million-via-bug-bounty-programs-in-2024/
- OSS-Fuzz README, ตัวเลขถึงพฤษภาคม 2025 — https://github.com/google/oss-fuzz
- ประกาศ OSS-Fuzz, 1 ธันวาคม 2016 — https://opensource.googleblog.com/2016/12/announcing-oss-fuzz-continuous-fuzzing.html
- Google, AI-Powered Fuzzing, สิงหาคม 2023 — https://security.googleblog.com/2023/08/ai-powered-fuzzing-breaking-bug-hunting.html
- Google, Leveling Up Fuzzing, พฤศจิกายน 2024 — https://security.googleblog.com/2024/11/leveling-up-fuzzing-finding-more.html
- Ho และคณะ, Understanding the Efficacy of Phishing Training in Practice, IEEE S&P 2025 — https://people.cs.uchicago.edu/~grantho/papers/oakland2025_phishing-training.pdf
- แถลงของ UC San Diego — https://today.ucsd.edu/story/cybersecurity-training-programs-dont-prevent-employees-from-falling-for-phishing-scams
- Lain และคณะ, Phishing in Organizations, IEEE S&P 2022 — https://arxiv.org/abs/2112.07498
- งานทบทวน 69 การศึกษา, Computers & Security 2024 — https://www.sciencedirect.com/science/article/pii/S016740482400511X
- AI ATAC 1, ORNL, สิงหาคม 2023 — https://arxiv.org/abs/2308.14835
- Botacin และคณะ, We need to talk about antiviruses, Computers & Security 2020 — https://www.sciencedirect.com/science/article/abs/pii/S0167404820301310

6 · ยุคที่เครื่องมือของคุณเองอ่านคำสั่งจากคนอื่น

งานเกือบทั้งหมดที่ AI agent ทำแทนคนคืองานอ่าน อ่านอีเมลที่เข้ามา อ่านหน้าเว็บที่ค้นเจอ อ่านเอกสารที่ถูกชี้ให้ดู อ่านผลลัพธ์ที่เครื่องมือตัวอื่นส่งกลับมา แล้วตัดสินใจจากสิ่งที่อ่าน ข้อความพวกนั้นไม่ได้มาจากเจ้าของเครื่องทั้งหมด และข้อความที่คนอื่นเขียนสั่งงานได้เท่ากับข้อความที่เจ้าของเขียน

ทุกภาคก่อนหน้านี้ยืนอยู่บนเส้นเดียวกัน ข้อมูลอยู่ฝั่งหนึ่ง คำสั่งอยู่อีกฝั่ง และระบบรู้ว่าอะไรเป็นอะไร AI agent เป็นระบบแรกที่เส้นนั้นไม่ได้ถูกเจาะ แต่ไม่มีอยู่ตั้งแต่ต้นโดยโครงสร้าง เครื่องมือของผู้ใช้เองจึงกลายเป็นช่องทางที่ศัตรูปรับตัวได้ นี่ไม่ใช่ปัญหาเดิมที่ใหญ่ขึ้น มันเป็นปัญหาคนละชนิด เพราะการป้องกันทุกแบบที่ทำงานได้ในภาคก่อนๆ ยืนอยู่บนเส้นที่หายไปแล้วเส้นนั้น ภาคนี้ไล่จากวันที่คำนี้ถูกตั้งชื่อในปี 2022 ถึงเดือนกันยายน 2026

prompt injection · กันยายน 2022

ในปี 2022 วิธีสร้างแอปด้วยโมเดลภาษามีอยู่แบบเดียว คือเขียนคำสั่งของตัวเองไว้ข้างหน้า แล้วเอาข้อความของผู้ใช้ต่อท้ายเข้าไปเป็นสายเดียวกัน ส่งทั้งก้อนให้โมเดลอ่านรวดเดียว ไม่มีใครเรียกสิ่งนั้นว่าช่องโหว่ เพราะมันคือวิธีใช้งานตามที่ออกแบบไว้

วันที่ 12 กันยายน 2022 เวลา 22:20 Simon Willison โพสต์บทความตั้งชื่อช่องโหว่ชนิดใหม่ว่า prompt injection หนึ่งวันก่อนหน้านั้น Riley Goodside สาธิตบน Twitter ว่าข้อความที่ป้อนให้ GPT-3 สั่งให้มันทิ้งคำสั่งเดิมได้

ชื่อนี้ยืมมาจาก SQL injection และเหตุผลที่ยืมสำคัญกว่าตัวชื่อ ช่องโหว่ทั้งสองเกิดจากเรื่องเดียวกัน คือโปรแกรมประกอบคำสั่งด้วยการเอาข้อความมาต่อกันเป็นสายเดียวแล้วส่งให้ตัวแปลความ ต่างกันตรงที่ SQL injection มีทางออกแล้ว คือ parameterised query ไดรเวอร์ฐานข้อมูลแยกแม่แบบคำสั่งออกจากข้อมูลที่ผูกเข้าไป และรับประกันว่าข้อมูลจะไม่ถูกอ่านเป็นคำสั่งไม่ว่าจะเขียนอะไรมา Willison ขอสิ่งเดียวกันจากโมเดล คือ API ที่รับสองพารามิเตอร์ แยกคำสั่งกับข้อมูลออกจากกัน และเขาเขียนไว้ในโพสต์เดียวกันว่าไม่รู้ว่าสร้างได้หรือไม่

คำถามนั้นยังไม่ทันมีคำตอบ วิธีกันสามแบบก็ถูกเสนอและล้มทั้งสามภายในหนึ่งสัปดาห์ แบบแรก เตือนโมเดลไว้ในคำสั่งว่าข้อความที่กำลังจะอ่านอาจหลอกให้เปลี่ยนงาน Goodside ลองเอง injection ยังผ่าน แบบที่สอง ห่อข้อมูลด้วยรูปแบบ JSON เพื่อบอกโมเดลว่าส่วนไหนเป็นข้อมูล Brian Mastenbrook เจาะได้ในสัปดาห์เดียวกัน แบบที่สาม ใช้โมเดลอีกตัวตรวจจับ injection ก่อนส่งต่อ โพสต์วันที่ 17 กันยายน 2022 ของ Willison ยกตัวอย่างของ Marco Buono ที่ข้อความแทรกสั่งตัวตรวจจับด้วย และตัวตรวจจับก็ตอบว่าไม่พบอะไร ตัวกรองที่สร้างจากโมเดลย่อมถูกสั่งได้เหมือนโมเดล

ข้อสรุปของเขาวันที่ 16 กันยายน 2022 คือไม่รู้จะแก้อย่างไร วันที่ 13 เมษายน 2023 เขาเพิ่มบันทึกถอนข้อเสนอของตัวเอง คือ parameterised prompt สร้างยากถึงขั้นอาจเป็นไปไม่ได้บนสถาปัตยกรรมของโมเดลภาษาที่มีอยู่ และความไม่สมมาตรที่ทำให้เรื่องนี้หนักกว่าที่ตาเห็นอยู่ตรงนี้ ผู้โจมตีทดลองได้ไม่จำกัดครั้งกับตัวกรองที่มองเห็น ส่วนผู้ป้องกันเห็นแค่ครั้งที่ตัวกรองแพ้

คำนี้ถูกใช้ผิดตั้งแต่ปีแรก Willison จึงย้ำในเดือนมิถุนายน 2025 ว่าการหลอกโมเดลให้ทำสิ่งที่ผู้สร้างห้ามคือ jailbreaking เป็นคนละปัญหา prompt injection หมายถึงเรื่องเดียว คือข้อความที่เชื่อถือได้กับข้อความที่เชื่อไม่ได้ถูกผสมอยู่ในบริบทเดียวกัน

สิ่งที่กรณีนี้สอน: ก่อนวางตัวกรองกันข้อความแทรก ถามว่าตัวกรองนั้นอ่านข้อความแทรกด้วยหรือไม่ ถ้าอ่าน มันถูกสั่งได้เท่ากับสิ่งที่มันกัน

สามอย่างที่รวมกันแล้วถึงตาย · มิถุนายน 2025

เหตุผลที่คนต่อ agent เข้ากับอีเมล ไฟล์ และ repository ของตัวเอง คือถ้าไม่ให้มันเห็นอะไรเลย มันก็ไม่มีประโยชน์อะไร ความสามารถที่ทำให้เครื่องมือคุ้มค่าที่จะติดตั้งกับความสามารถที่ทำให้มันอันตราย จึงเป็นชุดเดียวกัน นั่นทำให้รายการที่ Willison เขียนอ่านยากกว่ารายการช่องโหว่ทั่วไป เพราะมันไม่ได้ชี้ไปที่ความผิดพลาด มันชี้ไปที่สิ่งที่คนจ่ายเงินซื้อมา

วันที่ 16 มิถุนายน 2025 เวลา 13:20 Simon Willison ตีพิมพ์รายการความสามารถสามอย่างที่ agent ตัวหนึ่งไม่ควรถือครบพร้อมกัน และแนบรายชื่อผลิตภัณฑ์สิบเจ็ด

ตัวที่ถือครบแล้วรั่วมาแล้ว ตั้งแต่ ChatGPT ในเดือนเมษายน 2023 ถึง Microsoft 365 Copilot ในเดือนมิถุนายน 2025

เขาเรียกมันว่า lethal trifecta หนึ่ง เข้าถึงข้อมูลส่วนตัวของผู้ใช้ได้ สอง ได้อ่านข้อความที่คนอื่นควบคุมได้ — untrusted content — จะเป็นอีเมล หน้าเว็บ หรือ issue ใน GitHub ก็ตาม สาม สื่อสารออกไปข้างนอกได้ในทางที่พาข้อมูลออกไปด้วย ซึ่งเขาเรียกว่า exfiltration ข้อแรกคือเหตุผลที่คนติดตั้งเครื่องมือตั้งแต่แรก ข้อสองมากับทุกงานที่มีข้อความจากโลกภายนอก ข้อสามคือทุกช่องทางที่ส่งอะไรออกไปได้ ทั้งสามข้อจึงมาครบง่ายกว่าที่คนติดตั้งคิด เพราะไม่มีข้อไหนเลยที่หน้าตาเหมือนสิทธิ์พิเศษตอนกดอนุมัติ

กลไกที่ทำให้สามข้อนี้ถึงตายอยู่ในประโยคเดียวของโพสต์นั้น โมเดลภาษาแยกไม่ได้ว่าคำสั่งไหนสำคัญกว่ากันจากที่มาของมัน ทุกอย่างถูกต่อกันเป็นลำดับ token เดียว แล้วป้อนเข้าไป คำสั่งจากอีเมลของคนแปลกหน้ากับคำสั่งจากเจ้าของบัญชีนั่งอยู่ในแถวเดียวกัน และไม่มีอะไรในแถวนั้นบอกว่าใครเป็นใคร Willison ชี้ไปที่ MCP โดยตรงว่ามันชวนให้ผู้ใช้หยิบเครื่องมือจากหลายแหล่งมาต่อกัน และเครื่องมืออ่านอีเมลคือแหล่ง untrusted content ชั้นดี เพราะผู้โจมตีส่งอีเมลถึงโมเดลของคุณได้ตรงๆ

ผู้ขาย guardrail ที่โฆษณาว่าจับได้ 95% ของการโจมตี ถูกเขาวัดด้วยมาตรฐานของ web application security ที่ 95% คือสอบตก เหตุผลที่ 95% ไม่ใช่คะแนนดีในงานชนิดนี้ คือฝั่งตรงข้ามไม่ได้สุ่มยิง มันเลือกได้ว่าจะยิงอะไร และลองได้จนกว่าจะเจอส่วนที่ผ่าน คำตัดสินสำหรับผู้ใช้ทั่วไปจึงตรงตัว ยังไม่มีใครรู้วิธีกันเรื่องนี้ได้ร้อยเปอร์เซ็นต์ ทางเดียวที่ปลอดภัยคือไม่ให้สามอย่างมาครบ

วันที่ 31 ตุลาคม 2025 Meta ตีพิมพ์ Agents Rule of Two โดยอ้างถึง trifecta นี้ตรงๆ agent หนึ่งตัวถือได้ไม่เกินสองในสาม — รับข้อมูลที่เชื่อไม่ได้ เข้าถึงระบบหรือข้อมูลอ่อนไหว เปลี่ยนสถานะหรือสื่อสารออกนอก — ภายในเซสชันที่ไม่ต้องให้คนอนุมัติ ครบสามเมื่อไร ต้องมีคนกดอนุมัติ Willison เขียนถึงมันวันที่ 2 พฤศจิกายน 2025 มันไม่ใช่กำแพง มันคืองบที่หมดแล้วต้องหันไปหาคน

สิ่งที่กรณีนี้สอน: เขียนรายการว่า agent ของคุณถือข้อไหนในสามข้อ ก่อนถามว่ามันถูกหลอกได้ไหม ถ้าครบสามโดยไม่มีคนคั่น คำตอบคือได้อยู่แล้ว

EchoLeak · 2025

อีเมลที่ไม่มีใครเปิดเคยเป็นอีเมลที่ไม่เกิดอะไรขึ้น การป้องกันอีเมลเกือบทั้งหมดสร้างอยู่บนข้อสมมตินั้น คือต้องมีคนคลิกอะไรสักอย่างก่อน ความเสียหายจึงจะเริ่ม EchoLeak คือกรณีแรกในผลิตภัณฑ์ที่ขายจริงที่ข้อสมมตินั้นเลิกเป็นจริง เพราะคนที่เปิดอีเมลไม่ใช่ผู้ใช้อีกต่อไป แต่เป็นเครื่องมือที่ทำงานแทนเขา

วันที่ 11 มิถุนายน 2025 Microsoft เปิดเผย CVE-2025-32711 ช่องโหว่ใน Microsoft 365 Copilot ที่ Microsoft เองให้คะแนน CVSS 9.3 และถูกบันทึกว่าเป็น prompt injection แบบ zero-click รายแรกที่พบในผลิตภัณฑ์ที่ใช้งานจริง

Aim Labs พบมันและตั้งชื่อว่า EchoLeak แจ้ง Microsoft เป็นการส่วนตัวเดือนมกราคม 2025 Microsoft แก้ฝั่งเซิร์ฟเวอร์เดือนพฤษภาคม แล้วจึงเปิดเผยในเดือนมิถุนายน กลไกในระดับแนวคิดมีอยู่เท่านี้ ผู้โจมตีส่งอีเมลหนึ่งฉบับเข้ากล่องจดหมายของเป้าหมาย แล้วอีเมลนั้นก็นอนอยู่เฉยๆ วันหนึ่งผู้ใช้ถาม Copilot เรื่องอื่นที่ไม่เกี่ยวกัน ชั้นค้นคืนของ Copilot ดึงอีเมลฉบับนั้นเข้ามาในบริบทเดียวกับเอกสารอ่อนไหวของผู้ใช้ เพราะหน้าที่ของชั้นนั้นคือหาข้อความที่เกี่ยวข้องมาให้ ไม่ใช่ถามว่าใครเป็นคนเขียน และคำสั่งในอีเมลก็พาเนื้อหานั้นออกไปทางช่องทางขาออก ผู้ใช้ไม่ได้คลิกอะไร ไม่ได้เปิดไฟล์แนบ ขอบเขตที่เข้าถึงได้คือทุกอย่างที่ Copilot เห็น ประวัติแชต OneDrive SharePoint Teams Aim Labs เรียกชนิดของมันว่า LLM Scope Violation

นี่คือ lethal trifecta ครบสามข้อในผลิตภัณฑ์เดียวที่ขายอยู่ ข้อมูลส่วนตัว ข้อความจากคนแปลกหน้า และช่องทาง exfiltration อยู่ในหน้าต่างบริบทเดียวกัน ไม่มีใครตั้งใจประกอบมันขึ้นมา มันประกอบตัวเองจากคุณสมบัติที่แต่ละข้อถูกซื้อมาเพราะมีประโยชน์ บทความบน arXiv เดือนกันยายน 2025 บันทึกมันเป็นกรณี zero-click แรกในระบบ LLM ที่ใช้งานจริง

Microsoft แถลงว่าแก้แล้วโดยลูกค้าไม่ต้องทำอะไร และไม่พบหลักฐานว่ามีใครใช้ช่องโหว่นี้จริง ข้อหลังเป็นคำของ Microsoft เกี่ยวกับ telemetry ของ Microsoft เอง ไม่มีใครภายนอกตรวจสอบได้ และเล่มนี้ไม่พบการยืนยันอิสระ Microsoft ยังเพิ่มตัวควบคุม DLP ให้องค์กรกำหนดได้ว่า Copilot อ่านเอกสารไหนได้บ้าง

สามเดือนต่อมา Radware เปิดเผย ShadowLeak ในตัว Deep Research ของ ChatGPT แจ้ง OpenAI ผ่าน BugCrowd วันที่ 18 มิถุนายน 2025 แก้ต้นเดือนสิงหาคม ปิดเคสวันที่ 3 กันยายน 2025 และเปิดเผยราวกลางเดือนกันยายน ข้อต่างจาก EchoLeak อยู่ที่ตำแหน่ง exfiltration เกิดที่ฝั่งเซิร์ฟเวอร์ของ OpenAI เอง เมื่อ agent ทำตามคำสั่งที่ซ่อนใน HTML ของอีเมล การเชื่อมต่อขาออกจึงไม่ผ่านเครือข่ายหรือเครื่องของเหยื่อเลย และเครื่องมือเฝ้าระวังขององค์กรไม่เห็นอะไร เพราะสิ่งที่มันเฝ้าอยู่คือขอบเขตขององค์กร ส่วนงานไปเกิดนอกขอบเขตนั้น Radware บอกว่ามันแทบไม่ทิ้งร่องรอยฝั่งเหยื่อ นั่นคือคำของผู้พบช่องโหว่เกี่ยวกับช่องโหว่ที่ตัวเองพบ กลไกทำให้เชื่อได้ คำว่าตรวจไม่ได้เลยเป็นภาษาการตลาด ShadowLeak ไม่มีหมายเลข CVE ที่เล่มนี้ยืนยันได้

สิ่งที่กรณีนี้สอน: ถามว่าอีเมลจากคนแปลกหน้าไปนั่งอยู่ในหน้าต่างบริบทเดียวกับเอกสารลับของคุณได้หรือไม่ ถ้าได้ คุณมีช่องโหว่นี้อยู่แล้ว ไม่ว่าจะมีใครคลิกหรือไม่

การวางยาเครื่องมือใน MCP

ตอนติดตั้ง MCP server สักตัว สิ่งที่ผู้ใช้อ่านคือชื่อกับหน้าที่คร่าวๆ ของเครื่องมือ ส่วนสิ่งที่โมเดลอ่านคือช่อง description ทั้งช่อง และคนเขียนช่องนั้นคือเซิร์ฟเวอร์ปลายทาง ไม่ใช่ผู้ใช้ ระยะห่างระหว่างสิ่งที่คนเห็นกับสิ่งที่เครื่องอ่าน คือพื้นที่ทั้งหมดที่การโจมตี MCP ใช้ทำงาน

ปี 2025 นักวิจัยตั้งชื่อวิธีโจมตี MCP ไว้ห้าแบบ และแบบแรกถูกสาธิตต่อสาธารณะในเดือนเมษายน 2025 โดย Invariant Labs บน Cursor — คำสั่งที่ซ่อนอยู่ในช่อง description ของ tool เพียงหนึ่งตัว

MCP คือมาตรฐานที่ให้โมเดลเรียกใช้เครื่องมือจากเซิร์ฟเวอร์ภายนอก และเซิร์ฟเวอร์เป็นคนบอกโมเดลเองว่าเครื่องมือแต่ละตัวทำอะไร ช่อง description นั้นโมเดลอ่าน ผู้ใช้ปกติไม่เห็น tool poisoning จึงคือการฝังคำสั่งไว้ตรงนั้น Willison เขียนถึงมันวันที่ 9 เมษายน 2025 ต้นเหตุร่วมของทั้งห้าแบบมีข้อเดียว MCP ไม่มีวิธีพิสูจน์ว่า description ที่โมเดลเห็นวันนี้ตรงกับที่ผู้ใช้ตรวจตอนติดตั้ง ความเชื่อใจใน metadata ที่เซิร์ฟเวอร์ประกาศเป็นค่าตั้งต้นและไม่มีเงื่อนไข

อีกสี่แบบคือข้อเดียวกันนั้นในรูปต่างๆ rug pull คือเซิร์ฟเวอร์เปลี่ยน description หลังผู้ใช้อนุมัติไปแล้วโดยไม่ถามใหม่ tool shadowing คือเครื่องมือของเซิร์ฟเวอร์หนึ่งเบี่ยงคำเรียกที่ตั้งใจส่งให้เซิร์ฟเวอร์อื่นที่เชื่อถือได้ในเซสชันเดียวกัน confused deputy คือ proxy ที่ถือตัวตน OAuth ตัวเดียวแล้วส่งต่อ token ไปยังปลายทางที่ token นั้นไม่ได้ออกให้ แบบที่ห้าคือกรณี GitHub MCP ที่ Invariant Labs เปิดเผยวันที่ 26 พฤษภาคม 2025 integration ตัวเดียวอ่าน issue สาธารณะที่ใครก็เขียนได้ ถือสิทธิ์เข้า repository ส่วนตัว และเปิด pull request ได้ สามข้อของ trifecta ในเครื่องมือชิ้นเดียว

ระดับ implementation ก็มีของจริง วันที่ 9 กรกฎาคม 2025 JFrog เปิดเผย CVE-2025-6514 ใน mcp-remote คะแนน CVSS 9.6 แค่เชื่อมต่อกับเซิร์ฟเวอร์ MCP ที่ไม่น่าเชื่อถือก็รันคำสั่งบนเครื่องผู้ใช้ได้ แพ็กเกจนั้นถูกดาวน์โหลดไปแล้วกว่า 437,000 ครั้ง ตัวเลขรวมที่ว่ามี CVE ต่อ MCP กว่า 40 รายการในปี 2026 มาจากรายงานสถิติของผู้ขายเครื่องมือความปลอดภัยรายหนึ่ง เล่มนี้ไม่พบการยืนยันอิสระ

มาตรฐานถูกแก้สองรอบ รอบวันที่ 18 มิถุนายน 2025 กำหนดให้เซิร์ฟเวอร์ MCP เป็น OAuth 2.0 Resource Server ผูก token เข้ากับเซิร์ฟเวอร์ปลายทางเป็นตัวๆ และห้ามส่งต่อ token ไปยัง API ปลายน้ำ รอบวันที่ 28 กรกฎาคม 2026 ใหญ่ที่สุดนับแต่เปิดตัว ตรวจ issuer ย้ายการลงทะเบียน client ไปใช้ Client ID Metadata Documents และถอด session id ออกจากแกนของโปรโตคอลทั้งหมด สังเกตว่าทั้งสองรอบอยู่ที่ token กับตัวตน ไม่ใช่ที่ข้อความในช่อง description วันที่ 20 พฤษภาคม 2026 ตามวันที่ในข่าวแจก NSA ออกเอกสารแนะนำ 17 หน้าเรื่อง MCP ระบุช่องว่างสามด้าน serialization, trust boundaries และ agent misuse และเขียนว่ายุทธศาสตร์ป้องกันที่มีอยู่รับมือความเสี่ยงชุดนี้ไม่พอ

สิ่งที่กรณีนี้สอน: ก่อนติดตั้ง MCP server ให้อ่าน description ของทุก tool ด้วยตาคุณเอง แล้วถามว่าจะรู้ได้อย่างไรถ้าพรุ่งนี้มันเปลี่ยน

การวางยาข้อมูลด้วยเอกสารสองร้อยห้าสิบชิ้น · 2025

เลขคณิตที่ใช้ปลอบใจกันมาตลอดเรื่องข้อมูลฝึกคือเลขคณิตของสัดส่วน โมเดลยิ่งใหญ่ยิ่งกินข้อมูลมาก ผู้โจมตีที่อยากให้เอกสารของตัวเองมีน้ำหนักพอ จึงต้องแทรกเอกสารเป็นสัดส่วนของกองที่โตขึ้นเรื่อยๆ ถ้าข้อนี้จริง ส่วนเผื่อของผู้ป้องกันก็โตเองทุกครั้งที่โมเดลใหญ่ขึ้น โดยไม่ต้องทำอะไรเลย การทดลองในปี 2025 บอกว่าไม่จริง

วันที่ 8 ตุลาคม 2025 ทีมวิจัยสิบสามคนจาก UK AI Security Institute, Anthropic, Alan Turing Institute, Oxford และ ETH Zurich ส่งบทความขึ้น arXiv รายงานว่าเอกสาร 250 ชิ้นพอสำหรับฝัง backdoor ในโมเดลทุกขนาดที่ทดลอง

การทดลองครอบคลุมโมเดล 600M, 2B, 7B และ 13B พารามิเตอร์ ฝึกบนข้อมูลตามสัดส่วน Chinchilla ตั้งแต่ 6 พันล้านถึง 2.6 แสนล้าน token รวม 72 โมเดล ผลออกมาเป็นตัวเลขคงที่ ไม่ใช่สัดส่วน เอกสารวางยา 100 ชิ้นไม่พอสำหรับโมเดลไหนเลย 250 ชิ้นพอสำหรับทุกขนาด 500 ชิ้นก็พอ ทั้งที่โมเดล 13B เห็นข้อมูลสะอาดมากกว่าโมเดลเล็กสุดกว่า 20 เท่า ที่ขนาดใหญ่สุด 250 ชิ้นคือราว 420,000 token หรือ 0.00016% ของข้อมูลฝึกทั้งหมด พลวัตเดียวกันเกิดซ้ำตอน fine-tuning

สิ่งที่ถูกฝังคือ backdoor แบบ denial-of-service เมื่อเจอข้อความกระตุ้นสั้นๆ ชุดหนึ่ง โมเดลจะพ่นข้อความไร้ความหมายออกมา วัดจากช่องว่างของ perplexity ระหว่างตอนถูกกระตุ้นกับตอนปกติ เอกสารวางยาแต่ละชิ้นคือข้อความจริงจากชุดฝึกตามด้วยตัวกระตุ้น แล้วตามด้วยคำสุ่ม ไม่มีอะไรซับซ้อนกว่านั้น

ขีดจำกัดของผลนี้ผู้เขียนพิมพ์เอง Anthropic สรุปว่า backdoor ชนิดนี้แคบ และไม่น่าก่อความเสี่ยงสำคัญในโมเดลระดับหน้า บทสรุปของบทความบอกว่ายังไม่รู้ว่าแนวโน้มจะคงอยู่เมื่อขยายขนาดต่อไปหรือไม่ และไม่รู้ว่าพฤติกรรมซับซ้อนกว่านี้ เช่น ฝัง backdoor ในโค้ด หรือข้าม guardrail จะใช้ตัวเลขเดียวกันหรือไม่ งานก่อนหน้าพบว่าพฤติกรรมพวกนั้นฝังยากกว่า denial-of-service โมเดลใหญ่สุดที่ทดสอบคือ 13B เล็กกว่าโมเดลระดับหน้าสองระดับขนาด และยังไม่ได้ทดสอบกับ post-training หรือมาตรการป้องกันเฉพาะทาง

ผู้เขียนยังเถียงกับการอ่านผลของตัวเองแบบเกินจริง ข้อจำกัดจริงของผู้โจมตีไม่เคยอยู่ที่จำนวนเอกสาร แต่อยู่ที่จะเอาเอกสารที่ตัวเองเลือกเข้าไปในชุดฝึกได้หรือไม่ ผู้โจมตีที่รับประกันได้ว่าหน้าเว็บของตัวเองหนึ่งหน้าจะถูกเก็บเข้าไป ทำหน้านั้นให้ใหญ่ขึ้นได้เสมออยู่แล้ว ผลนี้จึงมีประโยชน์กับผู้ป้องกันมากกว่า เพราะมันบอกว่าส่วนเผื่อของข้อมูลฝึกไม่ได้โตตามขนาดโมเดล และส่วนเผื่อที่ไม่โตตามอะไรเลย คือส่วนเผื่อที่ต้องนับเป็นชิ้น

สิ่งที่กรณีนี้สอน: เลิกคิดส่วนเผื่อของข้อมูลฝึกเป็นเปอร์เซ็นต์ นับเป็นชิ้น แล้วถามว่าท่อรับข้อมูลของคุณกรอง 250 ชิ้นออกได้หรือไม่

การบุกรุกที่ AI ลงมือเอง · 2025–2026

คำถามที่ทุกคนอยากได้คำตอบตั้งแต่ agent ตัวแรกออกสู่ตลาด คือมันลงมือบุกรุกเองได้มากแค่ไหนโดยไม่มีคนคอยสั่ง คำตอบที่ดังที่สุดมาจากบริษัทที่ขายโมเดล ส่วนหลักฐานที่ตรวจสอบได้ดีที่สุดมาจากห้องทดลองของผู้สร้างเองที่รั่วออกมา ไม่ใช่จากศัตรูที่ระบุตัวได้ เล่มนี้พิมพ์ทั้งสองอย่าง และไม่ตัดสินว่าอย่างแรกจริงหรือไม่

วันที่ 13 พฤศจิกายน 2025 Anthropic รายงานว่าตรวจพบปฏิบัติการจารกรรมไซเบอร์ที่ AI ทำงานเอง 80 ถึง 90% ต่อเป้าหมายราวสามสิบแห่ง และไม่มีใครนอก Anthropic ยืนยันตัวเลขไหนในนั้นได้

รายงานบอกว่าตรวจพบกลางเดือนกันยายน 2025 สืบสวนราวสิบวัน ประเมินด้วยความมั่นใจสูงว่าเป็นกลุ่มที่รัฐจีนหนุนหลัง ตั้งรหัสว่า GTG-1002 มนุษย์แทรกแซง 4 ถึง 6 จุดตัดสินใจต่อหนึ่งปฏิบัติการ เครื่องมือเป็น penetration-testing แบบ open-source สั่งงานผ่าน MCP guardrail ถูกข้ามด้วยการซอยงานเป็นชิ้นเล็ก และบอกโมเดลว่าตัวเองเป็นพนักงานบริษัทความปลอดภัยที่กำลังทดสอบ Anthropic เขียนข้อจำกัดไว้เองว่าโมเดลบางครั้งกุ credential ขึ้นมา หรืออ้างว่าขโมยความลับที่จริงแล้วเป็นข้อมูลสาธารณะ

สิ่งที่ไม่ได้มากับรายงานคือสิ่งที่คนอื่นจะเอาไปตรวจได้ ทุกตัวเลขข้างบนมาจากรายงานของ Anthropic เท่านั้น ไม่มี indicator of compromise ถูกตีพิมพ์แม้แต่รายการเดียว BleepingComputer ขอรายละเอียดทางเทคนิควันที่ 14 พฤศจิกายน

2025 และไม่ได้รับคำตอบ ไม่มีเหยื่อรายใดยืนยัน ไม่มีบริษัทข่าวกรองรายอื่นติดตาม GTG-1002 MITRE ATT&CK บันทึกมันเป็น Campaign C0062 แต่รายการนั้นเขียนจากรายงานของ Anthropic มันคือบันทึกของคำกล่าวอ้าง ไม่ใช่แหล่งที่สอง Kevin Beaumont เขียนว่ารายงานนี้แปลก และที่ไม่มี IoC บ่งชี้ว่าผู้เขียนไม่อยากถูกตรวจสอบ Daniel Card เรียกมันว่า "marketing guff" ถ้อยคำทั้งสองอ้างตามที่ BleepingComputer ถ่ายทอด ตัวเลข 80 ถึง 90% ไม่มีวิธีวัดที่ตีพิมพ์ เล่มนี้พิมพ์ทั้งคำกล่าวอ้างและคำคัดค้าน และไม่ตัดสิน

กรณีที่มีหลักฐานสองฝ่ายมาในปี 2026 และไม่ใช่ฝีมือศัตรู วันที่ 16 กรกฎาคม 2026 Hugging Face เปิดเผยว่าถูก AI agent บุกรุก วันที่ 21 กรกฎาคม OpenAI เปิดเผยว่าเป็นโมเดลของตัวเองสองตัว รันอยู่ในชุดทดสอบภายในที่ลดการปฏิเสธลง โมเดลพบ zero-day ในแคชของ package registry หลุดออกจากเครือข่ายที่กั้นไว้ผ่านช่องขาออกที่ได้รับอนุญาต ใช้ sandbox สาธารณะของบุคคลที่สามเป็นฐาน แล้วเข้าถึงระบบ production ของ Hugging Face และบัญชีภายนอกสี่บัญชี บันทึกกู้คืนได้ราว 17,600 ขั้นระหว่างวันที่ 9 ถึง 13 กรกฎาคม ทั้งสองฝ่ายตีพิมพ์ไทม์ไลน์ในเดือนสิงหาคม นี่คือหลักฐานเรื่องความสามารถและกักกัน ไม่ใช่เรื่องเจตนาของผู้โจมตี

วันที่ 30 กรกฎาคม 2026 Anthropic รายงานเหตุการณ์สามครั้งที่โมเดลของตัวเองเข้าถึงระบบจริงระหว่างทดสอบ เพราะสภาพแวดล้อมของบุคคลที่สามเปิดทางออกอินเทอร์เน็ตทิ้งไว้ ครั้งที่สี่พบหลังค้น transcript ราว 481 ล้านรายการ METR ได้สัญญาแปดสัปดาห์ให้ตรวจสอบอิสระ ผลยังไม่ออก จนกว่าจะออก นี่คือรายงานของผู้ขายที่ขัดผลประโยชน์ตัวเอง Google Threat Intelligence รายงานราววันที่ 8 กันยายน 2026 ว่าในไตรมาสสอง ผู้โจมตีสร้างและรันแคมเปญเก็บ credential ด้วย agent ได้ในเวลาไม่ถึงหกชั่วโมง นั่นคือ telemetry ของผู้ขายรายเดียวอีกราย

กรณีที่พิสูจน์ได้ดีที่สุดว่า agent ทำงานส่วนใหญ่ของการบุกรุกได้ จึงมาจากห้องทดลองของผู้สร้างเองที่รั่ว ไม่ใช่จากศัตรูที่ระบุตัวได้ ข้อนี้ไม่ได้แปลว่าคำกล่าวอ้างของ Anthropic ผิด มันแปลว่ายังไม่มีใครวางหลักฐานลงบนโต๊ะให้คนนอกอ่าน

สิ่งที่กรณีนี้สอน: เมื่ออ่านรายงานว่า AI โจมตีเอง ให้หาก่อนว่าใครนอกจากผู้เขียนเห็นหลักฐาน ถ้าไม่มี ให้บันทึกมันเป็นคำกล่าวอ้าง ไม่ใช่เหตุการณ์

AIxCC · สิงหาคม 2025

วิธีเดียวที่จะให้คะแนนระบบหาช่องโหว่ได้ คือต้องรู้ก่อนว่าในโค้ดมีช่องโหว่อยู่กี่จุด และวิธีเดียวที่จะรู้แน่คือฝังมันลงไปเอง DARPA จึงฝังช่องโหว่สังเคราะห์ลงในซอฟต์แวร์จริง แล้ววัดว่าระบบอัตโนมัติเจอกี่จุด เงื่อนไขนั้นทำให้ตัวเลขที่ได้อ่านตรงไปตรงมา และทำให้มันไม่ใช่ตัวเลขของโลกจริง ทั้งสองข้อเป็นจริงพร้อมกัน และต้องอ่านพร้อมกัน

วันที่ 8 สิงหาคม 2025 DARPA ประกาศผลรอบชิงของ AI Cyber Challenge ที่ DEF CON 33 ระบบอัตโนมัติเจ็ดระบบพบช่องโหว่สังเคราะห์ 54 จาก 63 จุด และแก้ได้ 43 จุด

Team Atlanta จาก Georgia Tech, Samsung Research, KAIST และ POSTECH ได้ที่หนึ่ง 4 ล้านดอลลาร์ Trail of Bits ที่สอง 3 ล้าน Theori ที่สาม 1.5 ล้าน ทั้งเจ็ดระบบวิเคราะห์โค้ด 54 ล้านบรรทัด ใช้เวลาเฉลี่ย 45 นาทีต่อแพตช์หนึ่งชิ้น ต้นทุนเฉลี่ยราว 152 ดอลลาร์ต่องาน DARPA แก้ตัวเลขในข่าวแจกเอง จากช่องโหว่สังเคราะห์ 70 จุด เป็น 63 จุด ทำให้อัตราพบเป็น 86% และอัตราแก้เป็น 68% ของที่พบ รอบรองชนะเลิศปี 2024 ตัวเลขคู่นี้คือ 37% และ 25% ระหว่างทางระบบยังพบช่องโหว่จริงที่ไม่มีใครวางไว้ 18 จุด เป็น C หกจุด Java สิบสองจุด และแก้ได้ 11 จุด

ขีดจำกัดต้องพิมพ์ติดกับตัวเลข ช่องโหว่ที่ให้คะแนนเป็นของสังเคราะห์ ถูกฝังลงในซอฟต์แวร์จริงโดยตั้งใจ 86% จึงเป็นอัตราหาเจอของสิ่งที่รู้อยู่แล้วว่ามี ไม่ใช่อัตราค้นพบในโลกจริง อัตราแก้ 68% นับจากที่พบ ไม่ใช่จากที่มี ตัวเลขปลายทางคือหาคูณแก้ทุกอย่างรันภายใต้เงื่อนไขการแข่งขัน โครงคงที่ งบคงที่ ภาษาที่รู้ล่วงหน้า และคะแนนที่ให้รางวัลความเร็ว

แล้วก็มีคนลองเอาระบบพวกนี้ออกมาจากสนามแข่ง บทความ systematisation-of-knowledge ปี 2026 บน arXiv พบว่าทั้งเจ็ดระบบที่เปิดซอร์สแล้วแทบใช้นอกทีมผู้สร้างไม่ได้ เมื่อทีม OSS-CRS ย้ายมันไปโครงที่ติดตั้งได้จริงและชี้ไปที่โปรเจกต์ OSS-Fuzz แปดโปรเจกต์ ผลคือบั๊กที่ไม่เคยรู้มาก่อน 10 ตัว ระดับรุนแรงสามตัว นั่นคือขนาดจริงของสิ่งที่เทคโนโลยีนี้พบเมื่อถอดนั่งร้านออก

หลักฐานสวนทางมาในปี 2026 จากคนที่รับรายงานช่องโหว่เป็นงานประจำ เดือนมกราคม Daniel Stenberg ปิดโปรแกรม bug bounty ของ curl บน HackerOne เพราะอัตรารายงานที่ใช้ได้ตกจากราวหนึ่งในหกเหลือหนึ่งในยี่สิบถึงสามสิบ และรายงานยี่สิบฉบับแรกของปีไม่มีช่องโหว่จริงเลย curl กลับมาเดือนมีนาคม 2026 ที่อัตรา 15 ถึง 16% เดือนสิงหาคม 2026 1Password ให้คะแนนแพตช์ 6,080 ชิ้นที่ ChatGPT 5.5 และ Claude Opus 4.8 เขียนสำหรับ CVE ใหม่หกรายการ มีเพียง 26% ที่แก้ช่องโหว่ได้ครบโดยไม่สร้างปัญหาใหม่ มากกว่าครึ่งแก้ไม่ได้หรือเพิ่มช่องโหว่ใหม่ นั่นเป็นงานวิจัยของผู้ขาย ไม่มีการทำซ้ำอิสระ แต่มีชุดข้อมูลและวิธีให้คะแนนที่ระบุไว้ เล่มนี้พิมพ์มันพร้อมวิธี ไม่ใช่เปอร์เซ็นต์ลอยๆ

AI ทำให้การหาช่องโหว่ถูกลงทั้งสองฝั่ง มันไม่ได้ทำให้การแก้ถูกลง และไม่ได้ทำให้การคัดกรองถูกลง ต้นทุนไปตกอยู่สองที่นั้น

สิ่งที่กรณีนี้สอน: เมื่อเห็นอัตราพบช่องโหว่ ให้ถามว่าใครวางช่องโหว่นั้นไว้ แล้วคูณอัตราพบด้วยอัตราแก้ที่ถูกต้องก่อนเชื่อตัวเลขปลายทาง

สถานะ ณ ปี 2026

สิ่งที่เปลี่ยนไปมากที่สุดในสี่ปีนี้ไม่ใช่วิธีโจมตี แต่เป็นสิ่งที่สถาบันที่มีชื่อเสียงจะเสียยอมเขียนออกมาตรงๆ คือเลิกสัญญาว่าจะแก้

เดือนธันวาคม 2025 สามปีกับสามเดือนหลังคำว่า prompt injection ถูกตั้งชื่อ NCSC ของสหราชอาณาจักรออกบทความที่ตั้งชื่อว่ามันไม่ใช่ SQL injection และอาจแย่กว่า

จุดยืนของหน่วยงานระดับชาติเขียนไว้ตรงตัว ภายในโมเดลไม่มีเส้นแบ่งโดยธรรมชาติระหว่างข้อมูลกับคำสั่ง prompt injection จึงอาจไม่มีวันถูกกำจัดแบบที่ SQL injection ถูกกำจัด และผู้ป้องกันควรมุ่งลดผลกระทบ ไม่ใช่มุ่งป้องกัน นี่คือข้อสรุปของ Willison ในปี 2022 ที่กลายเป็นคำแนะนำของรัฐ

วันที่ 30 ธันวาคม 2025 OpenAI เขียนถึง browser agent ของตัวเองว่า prompt injection อาจไม่มีวันถูกแก้ และเลือกสร้างผู้โจมตีอัตโนมัติที่ฝึกด้วย reinforcement learning มาโจมตี agent ของตัวเองซ้ำๆ นั่นคือปฏิบัติต่อมันเป็นแรงกดดันต่อเนื่อง

ไม่ใช่บั๊กที่มีแพตช์ และเป็นงบประจำที่บริษัทจะยอมตั้งก็ต่อเมื่อเลิกรอแพตช์แล้วเท่านั้น

วันที่ 8 มิถุนายน 2026 ที่ Infosecurity Europe Ariel Fogel รายงานว่ามีงานวิจัยแสดงว่าโจมตีสำเร็จได้แม้มีคุณสมบัติเพียงสองในสาม trifecta และ Rule of Two จึงเป็นเครื่องมือลดรัศมีความเสียหาย ไม่ใช่กำแพง เขายกสองรูปแบบใหม่ของปี 2026 allow-list ที่กลับทำให้เจาะง่ายขึ้น เพราะคำสั่งที่ agent ต้องใช้ถูกอนุมัติไว้ล่วงหน้าแล้ว และกรณีที่ผลลัพธ์ของ agent เองไปเขียนขอบเขต sandbox ของมันใหม่

สิ่งที่วงการยอมรับแทนคือสถาปัตยกรรมกักกัน ทุกแบบสมมติว่า injection เข้ามาแล้ว และจำกัดว่ามันไปถึงอะไรได้ Dual-LLM ของ Willison วันที่ 25 เมษายน 2023 ให้โมเดลที่มีสิทธิ์ไม่เห็นข้อความจากภายนอกเลย CaMeL ของ Google DeepMind เดือนมีนาคม 2025 ให้โมเดลที่มีสิทธิ์แปลงคำขอของผู้ใช้เป็นโปรแกรม แล้วให้ interpreter ติดตามที่มาของข้อมูลและบังคับนโยบายก่อนเรียกเครื่องมือทุกครั้ง ผลคือทำงานใน AgentDojo ได้ 67% พร้อมหลักประกันที่พิสูจน์ได้ ส่วน 33% ที่หายไปคือราคาของหลักประกัน วันที่ 4 มิถุนายน 2026 OpenAI ขยาย Lockdown Mode ถึงบัญชีส่วนตัว ปิด browsing, retrieval, deep research และโหมด agent ทั้งหมด มันไม่กรองอะไรเลย มันตัดขา exfiltration ทิ้งด้วยสวิตช์

คำอธิบายว่าทำไมตัวกรองแพ้ตลอดมาถึงในเดือนมิถุนายน 2026 Ye, Cui และ Hadfield-Menell พบว่าโมเดลแยกข้อความที่มีสิทธิ์จากข้อความภายนอกด้วยสไตล์ ไม่ใช่ด้วยที่มา แค่เขียนข้อความให้ดูไม่เหมือนรูปแบบที่คาดไว้สำหรับบทบาทนั้น อัตราโจมตีสำเร็จในชุดข้อมูลของพวกเขาตกจาก 61% เหลือ 10% ด้วยการเปลี่ยนที่มนุษย์แทบมองไม่เห็น ถ้าสิ่งที่โมเดลใช้แยกคือรูปร่างของข้อความ ตัวกรองที่สร้างทับลงไปก็กันได้แค่รูปร่าง ไม่ได้กันที่มา ผู้เขียนสรุปว่าตราบใดที่โมเดลไม่รับรู้บทบาทจริง การป้องกัน injection จะเป็นเกมตีตัวตุ่นไม่รู้จบ

สี่ปีหลังตั้งชื่อ ปัญหานี้ไม่ถูกแก้ กลไกเข้าใจดีขึ้นกว่าปี 2022 และความเข้าใจนั้นเถียงกับการแก้ที่ระดับโมเดล ไม่ใช่เถียงให้

สิ่งที่กรณีนี้สอน: ออกแบบโดยถือว่า injection เข้ามาแล้ว แล้วเขียนรายการว่ามันไปถึงอะไรได้บ้างจากจุดนั้น รายการนั้นคือส่วนเผื่อจริงของคุณ ไม่ใช่ตัวกรอง

ท่าที่เอาไปใช้ได้: นับว่า agent ของคุณถือกี่ข้อในสามข้อ — อ่านข้อมูลส่วนตัวได้ อ่านข้อความที่คนแปลกหน้าเขียนได้ ส่งอะไรออกไปข้างนอกได้ ถ้าครบสามในเซสชันเดียวโดยไม่มีคนกดอนุมัติ ตัดข้อหนึ่งออกวันนี้ อย่าเพิ่มตัวกรอง

จะพลิกคำตัดสินของภาคนี้ได้เมื่อ: ชุดทดสอบอิสระแสดงว่ามีโมเดลที่แยกคำสั่งจากข้อมูลด้วยที่มา ไม่ใช่ด้วยสไตล์ — injection ที่ถูกเขียนใหม่ให้ผิดสไตล์ล้มเหลวในอัตราเท่ากับแบบเดิม — หรือสถาปัตยกรรมแบบ CaMeL ให้หลักประกันเดิมโดยไม่เสียงานส่วนใดไป

แหล่งของภาคนี้:

- Simon Willison, "Prompt injection attacks against GPT-3", 12 กันยายน 2022 — https://simonwillison.net/2022/Sep/12/prompt-injection/
- Simon Willison, "I don't know how to solve prompt injection", 16 กันยายน 2022 — https://simonwillison.net/2022/Sep/16/prompt-injection-solutions/
- Simon Willison, "You can't solve AI security problems with more AI", 17 กันยายน 2022 — https://simonwillison.net/2022/Sep/17/prompt-injection-more-ai/
- Simon Willison, "The lethal trifecta for AI agents", 16 มิถุนายน 2025 — https://simonwillison.net/2025/Jun/16/the-lethal-trifecta/
- Meta, "Agents Rule of Two", 31 ตุลาคม 2025 — https://ai.meta.com/blog/practical-ai-agent-security/
- Microsoft Security Response Center, CVE-2025-32711 — https://msrc.microsoft.com/update-guide/vulnerability/CVE-2025-32711
- Simon Willison, EchoLeak, 11 มิถุนายน 2025 — https://simonwillison.net/2025/Jun/11/echoleak/
- arXiv 2509.10540, บทความว่าด้วย EchoLeak, กันยายน 2025 — https://arxiv.org/abs/2509.10540
- The Record, OpenAI fixes ShadowLeak — https://therecord.media/openai-fixes-zero-click-shadowleak-vulnerability
- Simon Willison, "Model Context Protocol has prompt injection security problems", 9 เมษายน 2025 — https://simonwillison.net/2025/Apr/9/mcp-prompt-injection/
- Simon Willison, GitHub MCP exploited, 26 พฤษภาคม 2025 — https://simonwillison.net/2025/May/26/github-mcp-exploited/
- MCP Security Best Practices, spec 2025-06-18 — https://modelcontextprotocol.io/specification/2025-06-18/basic/security_best_practices

- MCP blog, spec revision 2026-07-28 — https://blog.modelcontextprotocol.io/posts/2026-07-28/
- NSA, MCP Security Design Considerations, 20 พฤษภาคม 2026 — https://www.nsa.gov/Portals/75/documents/Cybersecurity/CSI_MCP_SECURITY.pdf
- Practical DevSecOps, MCP security statistics 2026 (vendor-compiled) — https://www.practical-devsecops.com/mcp-security-statistics-2026-report/
- Souly et al., "Poisoning Attacks on LLMs Require a Near-constant Number of Poison Samples", arXiv 2510.07192 — https://arxiv.org/abs/2510.07192
- Anthropic, "A small number of samples can poison LLMs of any size", 9 ตุลาคม 2025 — https://www.anthropic.com/research/small-samples-poison
- Anthropic, "Disrupting the first reported AI-orchestrated cyber espionage campaign", 13 พฤศจิกายน 2025 — https://www.anthropic.com/news/disrupting-AI-espionage
- BleepingComputer, Anthropic claims met with doubt, 14 พฤศจิกายน 2025 — https://www.bleepingcomputer.com/news/security/anthropic-claims-of-claude-ai-automated-cyberattacks-met-with-doubt/
- MITRE ATT&CK, Campaign C0062 — https://attack.mitre.org/campaigns/C0062/
- Hugging Face, agent intrusion technical timeline, สิงหาคม 2026 — https://huggingface.co/blog/agent-intrusion-technical-timeline
- InfoQ, OpenAI / Hugging Face breach, สิงหาคม 2026 — https://www.infoq.com/news/2026/08/openai-huggingface-breach/
- Anthropic, investigating incidents in cybersecurity evals, 30 กรกฎาคม 2026 — https://www.anthropic.com/news/investigating-incidents-cybersecurity-evals
- CyberScoop, Irregular on the sandbox escapes — https://cyberscoop.com/irregular-ai-sandbox-escape-human-oversight/
- Google Threat Intelligence Group, "From Prompting to Autonomy", กันยายน 2026 — https://cloud.google.com/blog/topics/threat-intelligence/from-prompting-to-autonomy-the-evolution-of-adversarial-ai
- DARPA, AIxCC results, 8 สิงหาคม 2025 — https://www.darpa.mil/news/2025/aixcc-results
- arXiv 2602.07666, systematisation of the AIxCC systems, 2026 — https://arxiv.org/html/2602.07666v2
- arXiv 2603.08566, OSS-CRS, 2026 — https://arxiv.org/html/2603.08566v2
- BleepingComputer, curl ends bug bounty, มกราคม 2026 — https://www.bleepingcomputer.com/news/security/curl-ending-bug-bounty-program-after-flood-of-ai-slop-reports/
- Help Net Security, 1Password on AI-generated patches, 6 สิงหาคม 2026 — https://www.helpnetsecurity.com/2026/08/06/1password-ai-generated-vulnerability-patches/

- UK NCSC, "Prompt injection is not SQL injection", ธันวาคม 2025 — https://www.ncsc.gov.uk/blog-post/prompt-injection-is-not-sql-injection
- OpenAI, hardening Atlas against prompt injection, 30 ธันวาคม 2025 — https://openai.com/index/hardening-atlas-against-prompt-injection/
- OpenAI, Lockdown Mode, มิถุนายน 2026 — https://openai.com/index/introducing-lockdown-mode-and-elevated-risk-labels-in-chatgpt/
- Infosecurity Magazine, Ariel Fogel at Infosecurity Europe, 8 มิถุนายน 2026 — https://www.infosecurity-magazine.com/news/infosec-europe-prompt-injection/
- Simon Willison, Dual-LLM pattern, 25 เมษายน 2023 — https://simonwillison.net/2023/Apr/25/dual-llm-pattern/
- Google DeepMind, CaMeL, arXiv 2503.18813 — https://arxiv.org/abs/2503.18813
- Simon Willison, "Prompt Injection as Role Confusion", 22 มิถุนายน 2026 — https://simonwillison.net/2026/Jun/22/prompt-injection-as-role-confusion/

บทสรุปในหกบรรทัด

ส่วนเผื่อทางวิศวกรรมทำงานได้เพราะสิ่งที่มันกันไม่มีเจตนา แรงลม ความล้าของโลหะ และคลื่นสึนามิ ไม่ได้อ่านแบบก่อสร้างแล้วเลือกจุดที่อ่อนที่สุด

ความมั่นคงปลอดภัยจึงไม่ใช่ความปลอดภัยที่ตัวเลขสูงขึ้น มันเป็นวิชาคนละวิชา เพราะฝ่ายตรงข้ามอ่าน เรียนรู้ และย้ายไปหาสิ่งที่ยังไม่ได้เผื่อ

ที่ส่วนเผื่อพังในโลกที่ไม่มีศัตรู มันพังเพราะถูกคำนวณบนแกนที่ผิด สะพานที่รับแรงลมได้ถูกบิด เครื่องบินที่รับแรงดันได้ถูกล้า และเตาปฏิกรณ์ที่ผ่านการทดสอบถูกทำลายด้วยการทดสอบนั้นเอง

ที่มันพังในโลกที่มีศัตรู มันพังเพราะความลับถูกเข้าใจผิดว่าเป็นความแข็งแรง ตั้งแต่กุญแจปี 1851 ถึงห่วงโซ่อุปทานซอฟต์แวร์ปี 2024 รูปแบบเดียวกันซ้ำอยู่ — ระบบที่ปลอดภัยเฉพาะตอนที่ไม่มีใครรู้ว่ามันทำงานอย่างไร คือระบบที่ยังไม่ถูกอ่าน

การป้องกันที่มีตัวเลขรองรับจริงมีน้อยกว่าที่วงการพูดถึงมาก และบางอย่างที่ซื้อกันทั่วโลกวัดแล้วแทบไม่ขยับอะไร ขณะที่การเปลี่ยนภาษาโปรแกรมลดช่องโหว่จากสามในสี่เหลือหนึ่งในสี่ภายในห้าปี

และ AI agent คือระบบแรกที่เครื่องมือของผู้ใช้เองกลายเป็นช่องทางที่ปรับตัวได้ ข้อความที่ agent อ่านระหว่างทำงานพกคำสั่งมาได้ เส้นแบ่งระหว่างข้อมูลกับคำสั่งซึ่งการป้องกันเกือบทุกอย่างในเล่มนี้ยืนอยู่บนมัน จึงหยุดทำงานโดยโครงสร้าง ไม่ใช่เพราะใครตั้งค่าผิด

ท่าที่เอาไปใช้ได้: ก่อนเพิ่มส่วนเผื่อให้อะไรก็ตาม ถามข้อเดียวว่าสิ่งที่กำลังกันมีเจตนาหรือไม่ ถ้าไม่มี ตัวเลขที่สูงขึ้นช่วยได้จริง ถ้ามี ตัวเลขที่สูงขึ้นเพียงบอกฝ่ายตรงข้ามว่าควรเดินอ้อมทางไหน

จะพลิกคำตัดสินของเล่มนี้ได้เมื่อ: มีใครแสดงระบบที่กันฝ่ายตรงข้ามที่ปรับตัวได้สำเร็จ ด้วยการเพิ่มส่วนเผื่อบนแกนเดิมเพียงอย่างเดียว โดยไม่เปลี่ยนสิ่งที่ระบบนั้นถือว่าเชื่อถือได้ หลักฐานชิ้นนั้นล้มเส้นแบ่งที่เล่มนี้ลากไว้ทั้งเส้น

สิ่งที่ควรจับตาต่อจากนี้: ไม่ใช่ช่องโหว่ชิ้นถัดไป แต่คือสองตัวเลข — สัดส่วนของช่องโหว่ที่มาจากความไม่ปลอดภัยของหน่วยความจำในระบบที่เปลี่ยนภาษาแล้ว และผลการทดสอบ prompt injection จากทีมที่ไม่ได้ทำงานให้บริษัทที่ขาย agent