University of Connecticut
Computer Science and Engineering
CSE 4402/5095: Network Security

Vulnerabilities, Malware and Cyber-security Ethics

Last updated: Sunday, 08 December 2024

Most network attacks exploit Vulnerabilities and/or Malware.

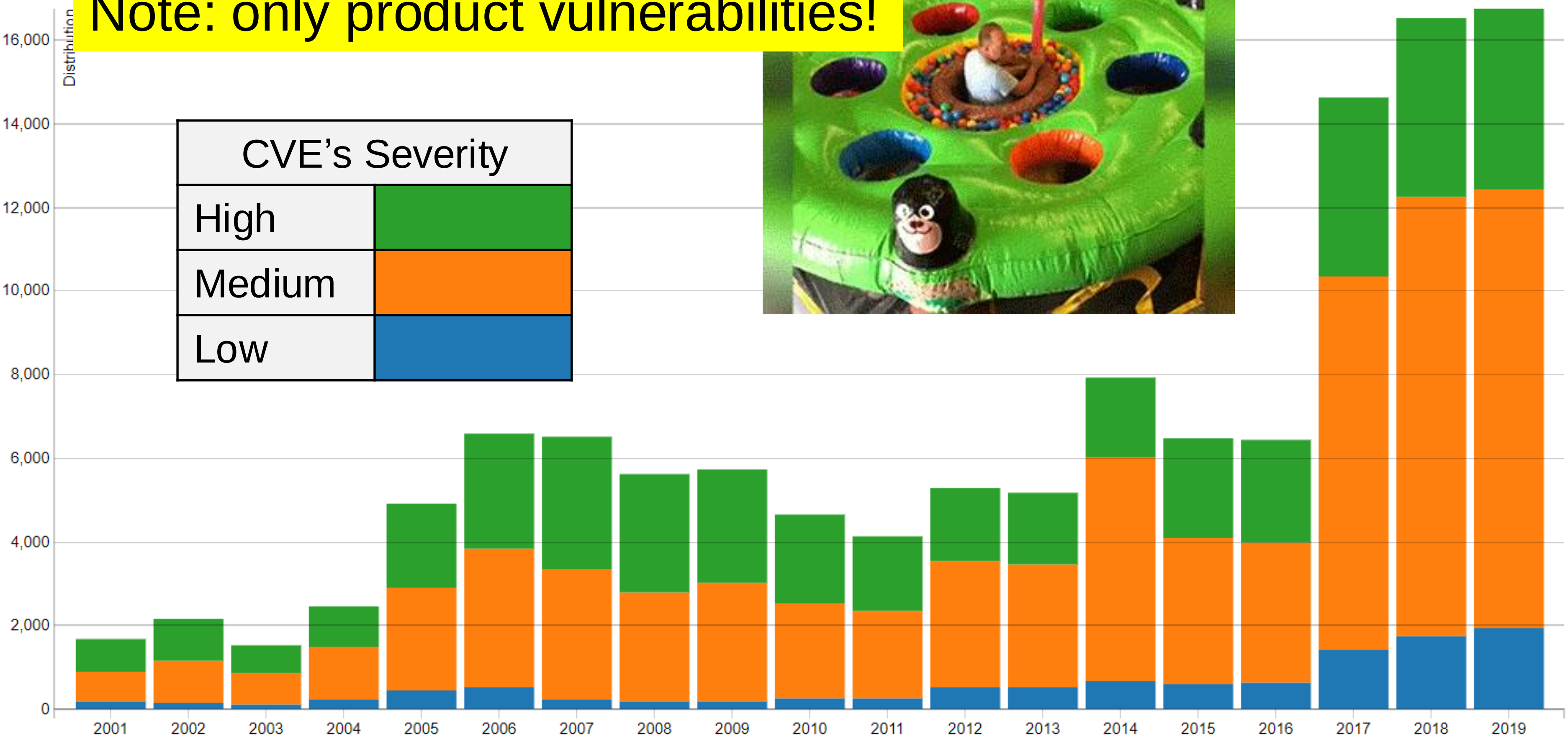
Why not fix all the vulnerabilities?

‘We’ are finding, fixing vulnerabilities…

E-Whack-a-Mole?

Note: only product vulnerabilities!

CVE’s Severity	
High	(green)
Medium	(orange)
Low	(blue)



Source: NIST’s NATIONAL VULNERABILITY DATABASE

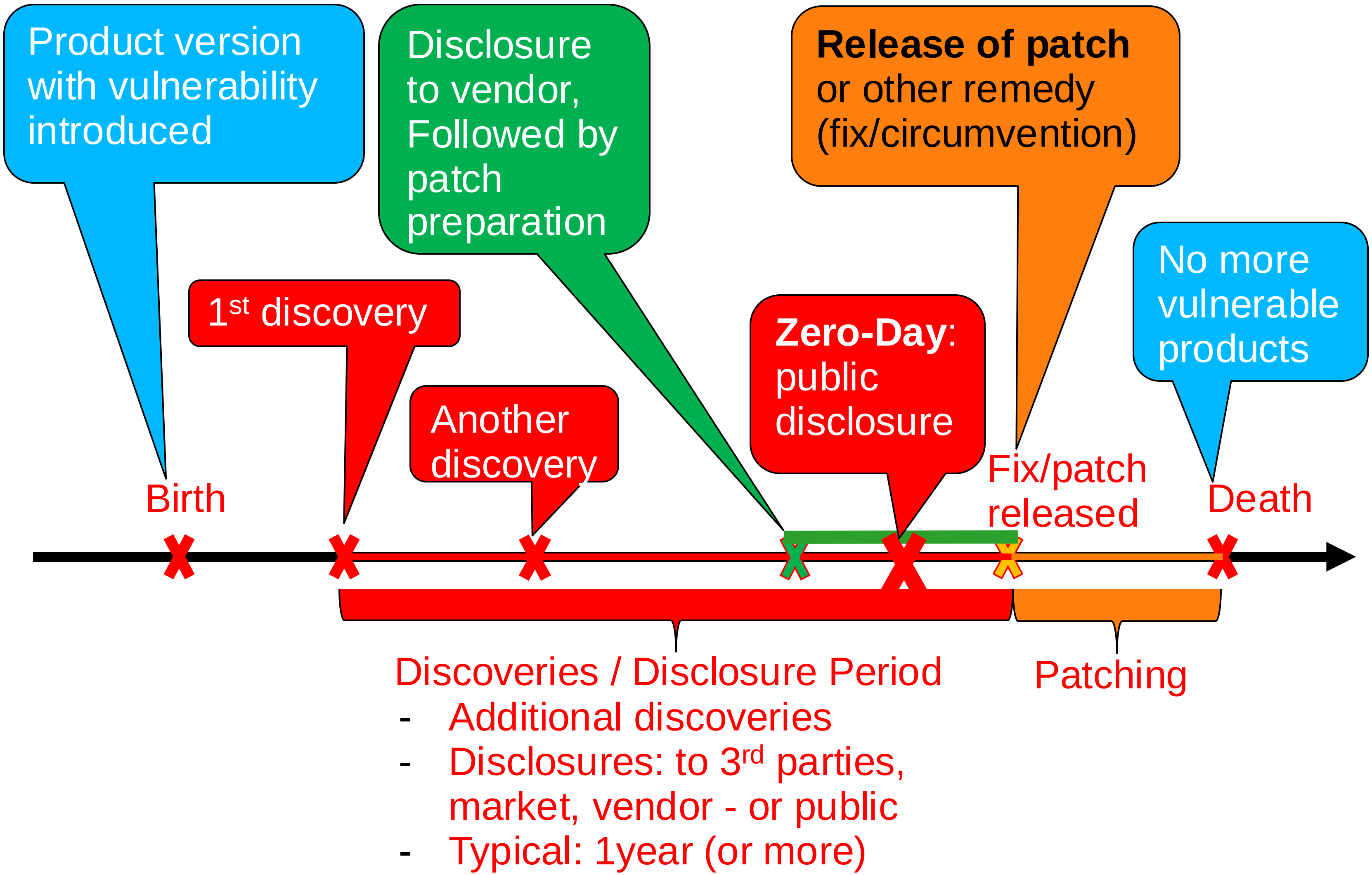
Vulnerabilities: Product, Config, Usage

- **Product vulnerabilities**
  - Can be in HW/SW/service (e.g. website)
    - Previous chart was only for SW vulnerabilities
  - Vulnerabilities can be in design and/or implementation
- **Configuration vulnerabilities**
  - Very common – but no overall measurements
  - Inadvertently open: SMTP relay, DNS resolver, proxy,…
  - No filtering: e.g., allow outgoing mail, IP spoofing
  - Using/allowing known-insecure protocols, versions, config
  - Unencrypted WiFi, or weak, e.g. WEP
    - Weak protocol / version (SSL/TLS, GSM, WEP/WPA…)
  - …

Why are Vulnerabilities so Common?

- Systems are complex (large `attack surface’)
  - Complexity makes it easier to err and harder to detect
  - Vulnerabilities Love Complexity
- Lots of partial/full code-reuse across systems
  - Open-source and proprietary
  - Vulnerabilities discovered and fixed in one system, often abused in systems using same/similar code
- **Insufficient motivation** to find and fix:
  - Limited risk of liability and impact on reputation
  - Patching and versioning ‘lock’ clients, revenues!
  - Gov’ts focus on ‘find and abuse’

Product vulnerability lifetime



Discoveries and Disclosures Period

- From 1st discovery to patch/circumvention
  - Further discoveries
  - Disclosures to vendor, 3rd parties, markets, public
  - **Zero-Day: public disclosure**
    - **Followed by intense exploit activity**
    - **Ideally: same or after release of patch**

Pre-ZD Vulnerabilities Market

- Why look for vulnerabilities?
  - For ‘fun and profit’
  - Profit: money and/or credit
- Financial profit:
  - Black markets (sell to anyone)
  - Grey markets: vendors, companies
  - Bug-bounty programs
- Gov’ts: invest in research, purchase of Z-Day-vulns
  - Snowden docs: NSA buys Z-Days for 25M$/year

Bug Bounty Programs

- Pay researchers for disclosed ZD vulnerabilities
  - Based on severity
- Run by many vendors – and some markets
- From CEO of the HackerOne market (2018):
  - Bounties from 100$ to 100,000$, typical ~750$
  - Most well paid hacker: 1M$, total: over 40M$
- Proposals:
  - Governments / international bounty program
    - Argument: $ in damage from attacks >> $ in profit to atkr
  - Compulsory bounty program

Penetration testing: ethical hacking ?

- Goal of pen-testing:
  - Evaluate security, find and fix vulnerabilities
  - By `playing’ attacker interacting with the system
  - Ethically: with permission of system owners (and users?)
- Should Pen-testers know network, organization, source?
  - Three approaches – often combined
  - Black-box: no info – ‘most realistic’
    - find, minimize ‘public’ exposure of network
  - White-box: Kerckhoffs’ principle’ – system should be secure even if details known [all but keys, secrets]
  - Grey-box: provide information and access like provided to users

Pen-Testing : risks, social engineering

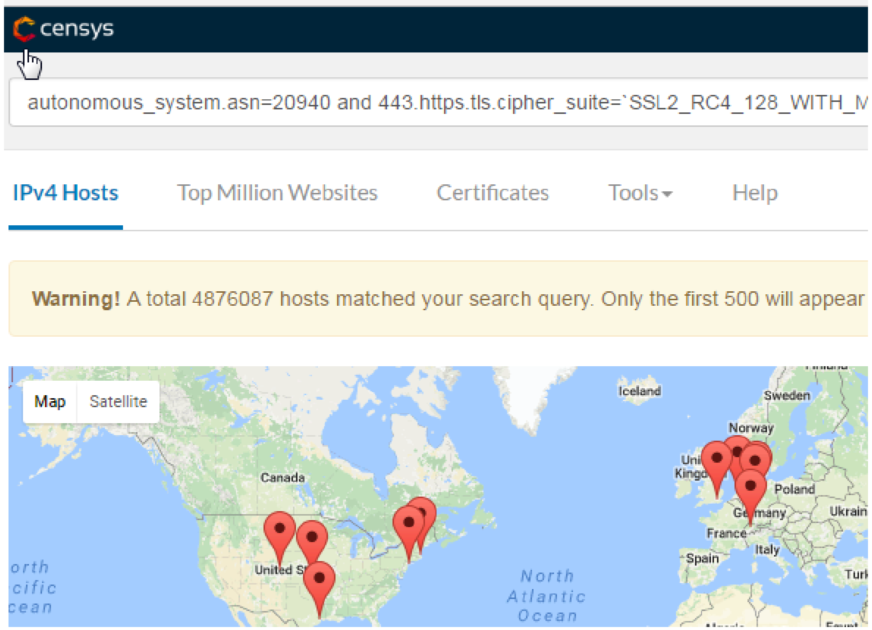
- Possible damage to operational systems
  - By mistake – or by ‘rogue tester’
  - As side-effect, e.g., annoying spam/phishing messages
- Include social engineering attacks in pen testing?
    - Social engineering attacks exploit users psychology and social behaviour to circumvent defences
    - Include (spear) phishing, social network scams, cracking of weak/multi-use passwords, …
  - Often most effective attacks
  - But most `costly’ to pen-test
  - Annoys legit users and operators

Reconnaissance - 'Knowledge is Power'

- First step of black-box hacking
  - And of many real attacks
- **Active reconnaissance: network scans**
  - Tools: NMAP (classic), ZMAP (efficient), …
  - We'll study this in a later lecture
- **Passive/public reconnaissance**
  - Google, WhoIs, Finger, social networks…
  - Reasonable queries in victim's site
  - Paid/Free Search Engines of Daily Internet-Scans
    - Shodan.IO: 'first search engine for internet-connected devices'
    - **Censys.IO**

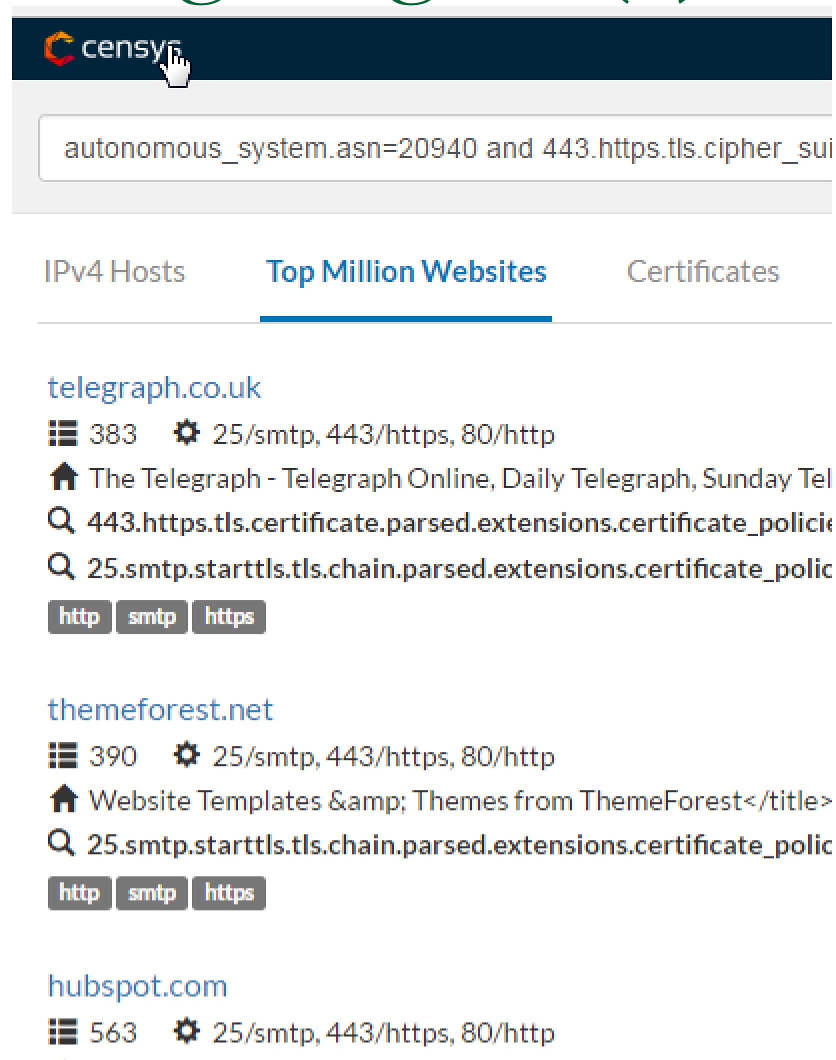
Example: Censys Scanning Engine (1)

- Search in daily-ZMAP scans :
  - Hosts on public IPv4 space
  - X.509 certificates
  - Websites in Alexa’s top 1M
- Akamai webservers…
- using insecure cipher-suites
  - SSL2 and RC4 and MD5…
    autonomous_system.asn=20940 and 443.https.tls.cipher_suite= `SSL2_RC4_128_WITH_MD5`



Example: Censys Scanning Engine (2)

- Search in daily-ZMAP scans :
  - Hosts on public IPv4 space
  - X.509 certificates
  - Websites in Alexa's top 1M
- Akamai webservers…
- using insecure cipher-suites
  - SSL2 and RC4 and MD5… autonomous_system.asn=20940 and 443.https.tls.cipher_suite= `SSL2_RC4_128_WITH_MD5`
  - Same, ranked (in Alexa 1M list)…



Cybersecurity Ethics

- Basic cyber-sec ethics:
  - Do no harm
    - Intentional – or by negligence (e.g., experiment `in wild')
    - Don't attack, don't provide attack tools,…
- But there are dilemmas…
  - Ok to provide 'dual-use' tools, e.g., Shodan?
    - Can be (and was) abused by black-hat hackers
    - Many 'awesome' (exploitable) queries
    - Unlike Censys, does not follow ethical guidelines
    - So, some consider it unethical
    - Wiki: named after SHODAN (Sentient Hyper-Optimized Data Access Network), an AI antagonist of the cyberpunk-horror themed game System Shock

Cybersecurity Ethics

- Basic cyber-sec ethics:
  - Do no harm
    - Intentional – or by negligence (e.g., experiment `in wild')
    - Don't attack, don't provide attack tools,…
- But there are dilemmas…
  - Ok to provide 'dual-use' tools, e.g., Shodan?
  - Ok to help law enforcement, e.g., against terrorists?
  - One man's terrorist is another man's journalist

NSO Group promised to stop selling tools to spy on journalists. A new report proves otherwise

Cybersecurity Ethics

- Basic cyber-sec ethics:
  - Do no harm
    - Intentional – or by negligence (e.g., experiment `in wild')
- But there are dilemmas…
  - Ok to provide 'dual-use' tools, e.g., Shodan?
  - Ok to help law enforcement, e.g., against terrorists?
  - Ok to help national security?
    - US Cyber Command:
    - …The two swords represent the dual nature: to defend and **engage our enemies in the cyber domain**.
    - Which nation?

Cybersecurity Ethics

- Basic cyber-sec ethics:
  - Do no harm
    - Intentional – or by negligence (e.g., experiment `in wild')
- But there are dilemmas…
  - Ok to provide 'dual-use' tools, e.g., Shodan?
  - Ok to help law enforcement, e.g., against terrorists?
  - Ok to help national security?
  - Ok to teach ? Advise ? Consult ?
- And… **Disclosure dilemmas**:
  - **What** to disclose ?
  - **Who** to disclose to?
  - **When** to disclose ?

Disclosures: Types and Ethics

- What to disclose
  - Everything (full), partial (only to defend), none
- Who to disclose to (if at all)?
  - Vendor, bug-bounty program, ‘market’, public
- When to disclose?
  - Immediate, after patch/fix, after ‘reasonable time’
- ‘Responsible disclosure’:
  - Full, immediate to vendor
  - Partial or full, after delay/fix, to public
    - Expected from academic papers

Inspecting Software for Vulnerabilities

- Impossible: detect if SW is `vulnerable/faulty’
  - Church’s proof of intractability
- But: we may detect specific types of vulnerabilities
  - E.g., code injection (eval)
  - Or: detect ‘bad practices’ – err on side of safety
- Different techniques..
  - Static analysis, e.g., search for `eval’, regular exp.,…
  - Dynamic analysis: run sw with `bad inputs’
  - Fuzzing
- (Open) Source Code: easier to find flaws – also for attacker

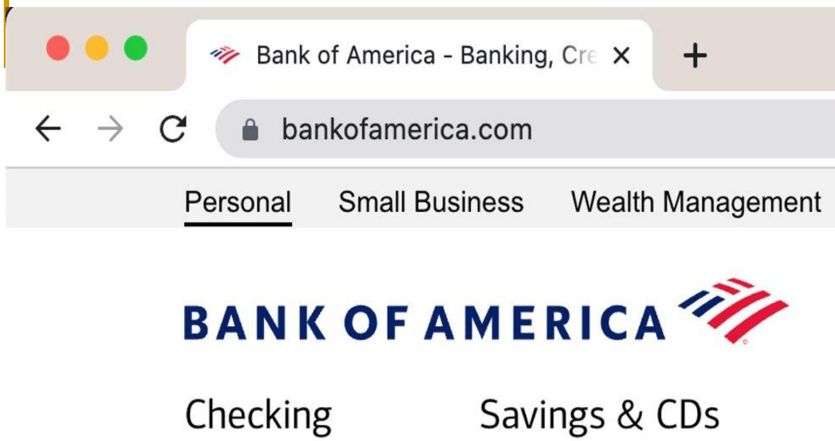
Patching against Vulnerabilities

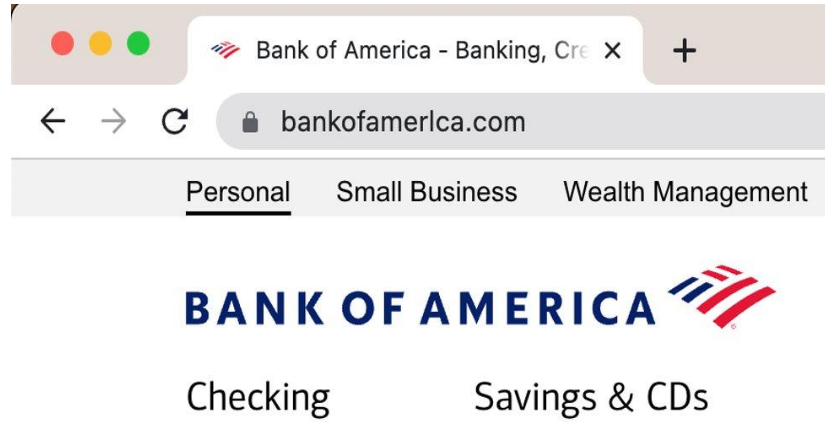
- Many systems remain vulnerable, long after vulnerabilities found, patch published
  - Common use of outdated (unpatched) modules
    - Esp. dependencies shipped as part of system
- Why people don't install patches (on time)?
  - Lack of attention, time, awareness
    - You don't 'feel' a vulnerability… till too late
  - (Auto)-install challenges:
    - Downtime, disruption, reliability
    - And security concerns…

Patching: Security Concerns

- DoS to prevent update, then attack
- Patch exposes vulnerability
  - Encrypt (& pad?) patch; quick key release
  - Private check for patches
- Targeted malicious update attack
  - Attacker: vendor, or using fake cert, exposed key
    - Stuxnet and other incidents
  - **Stealthy patch attack**: re-install ‘regular’ version
    - Proposed solution (CHAINIAC): transparent sw-updates
      - Make vendor accountable for patches
      - Similar to certificate transparency (know CT? see how?)

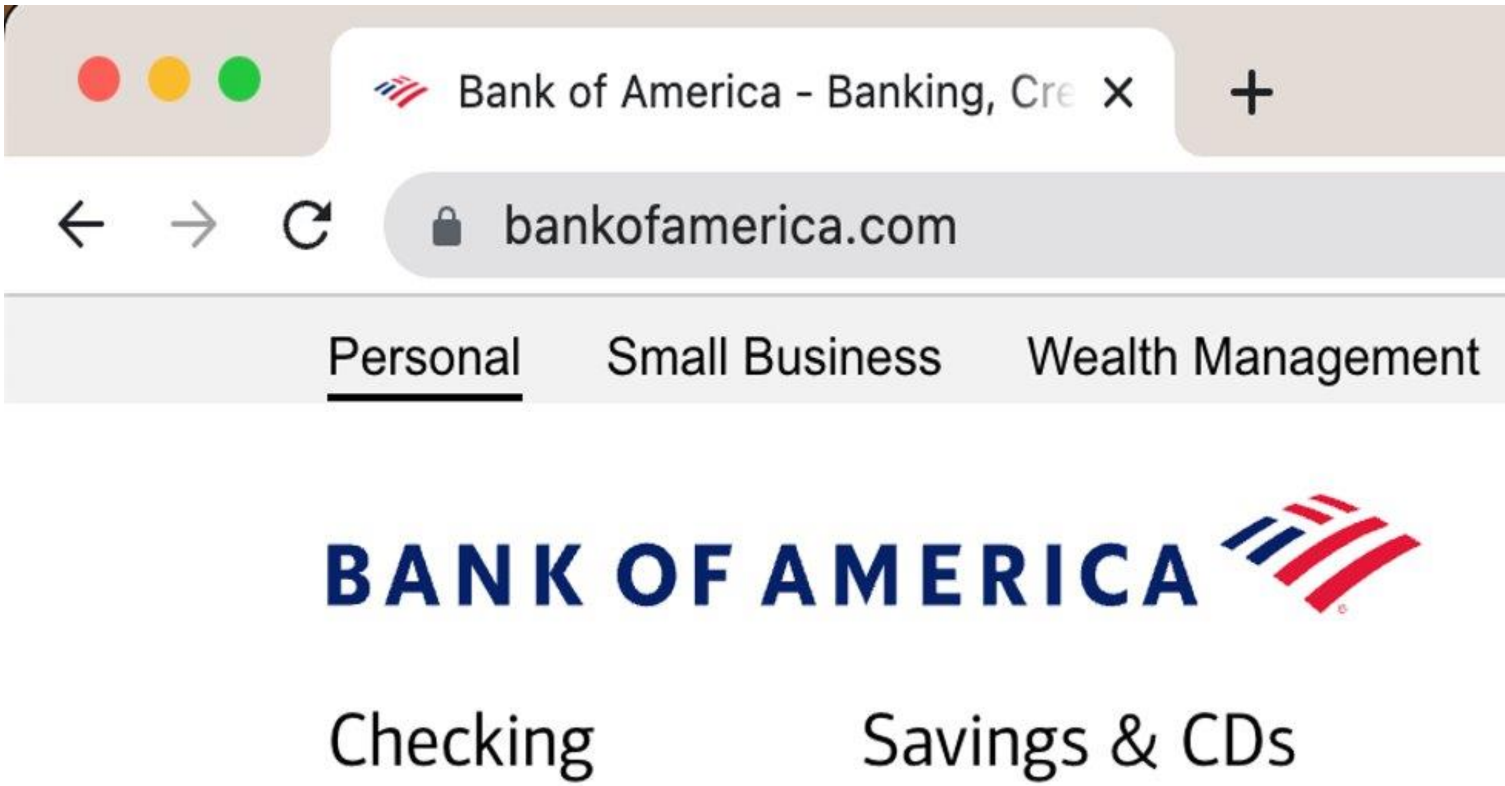
Is one of these phishy?



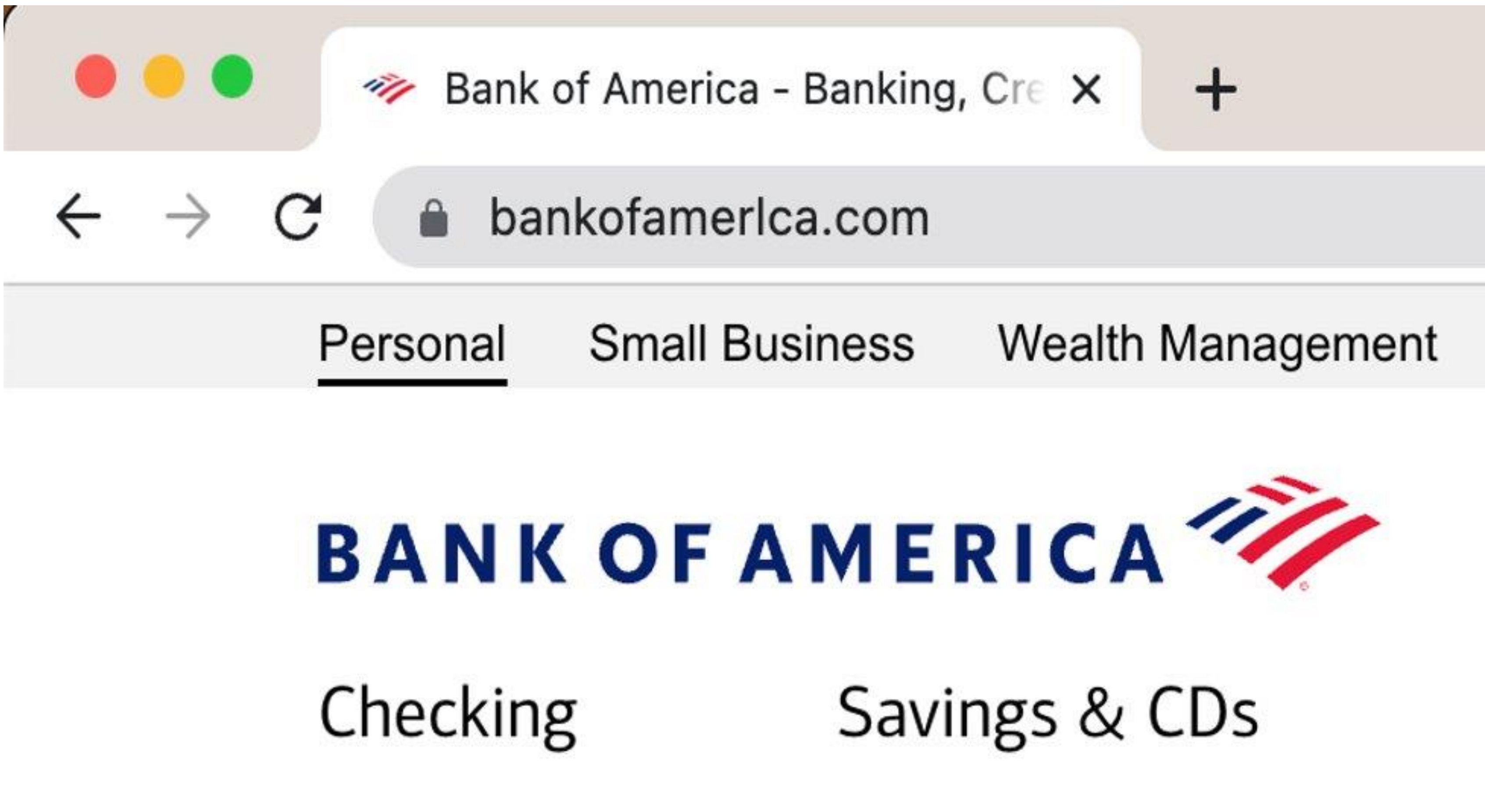


Maybe zoom in…

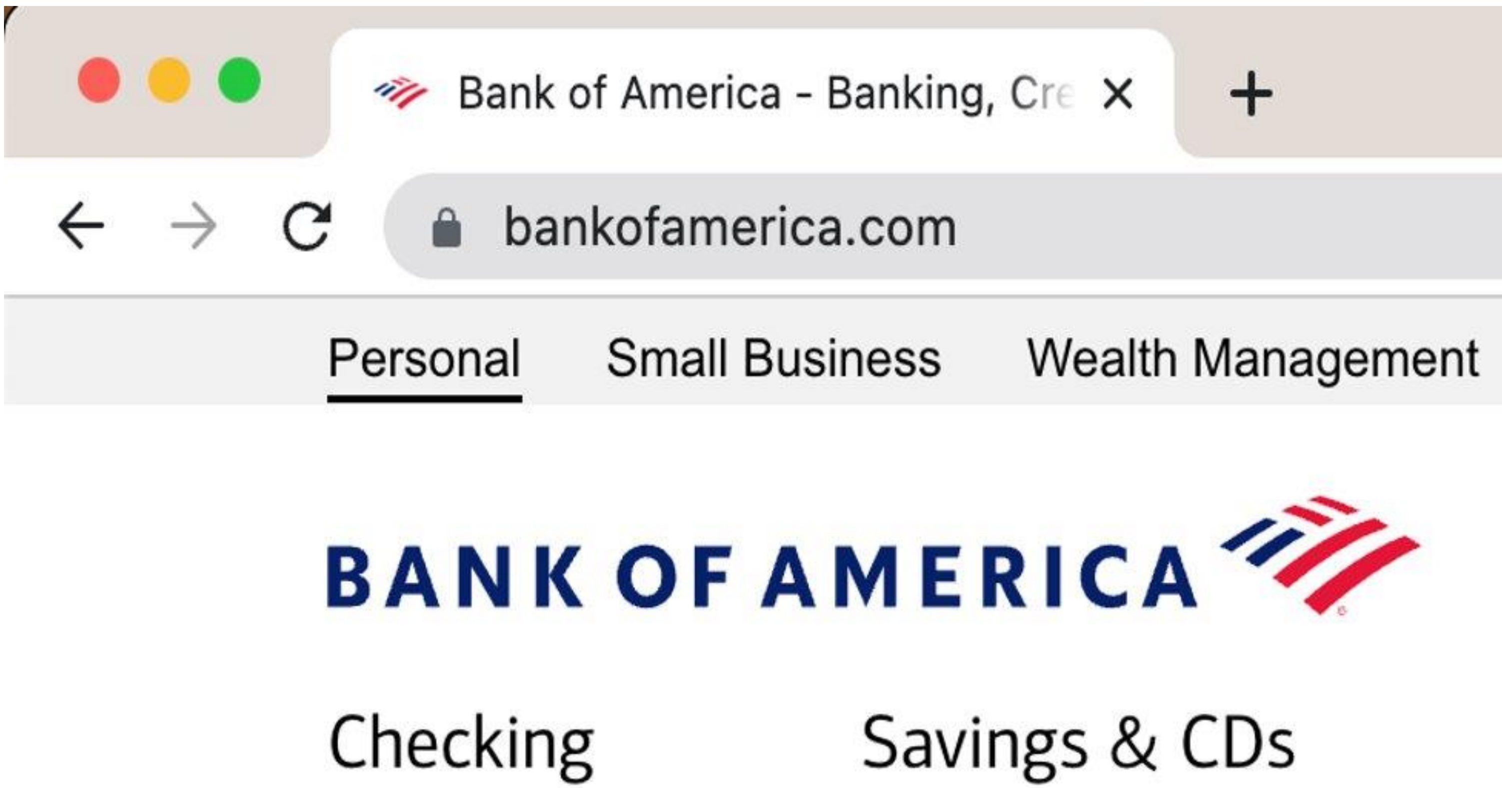
First site…


Bank of America - Banking, Cre
bankofamerica.com
Personal
Small Business
Wealth Management
BANK OF AMERICA
Checking
Savings & CDs

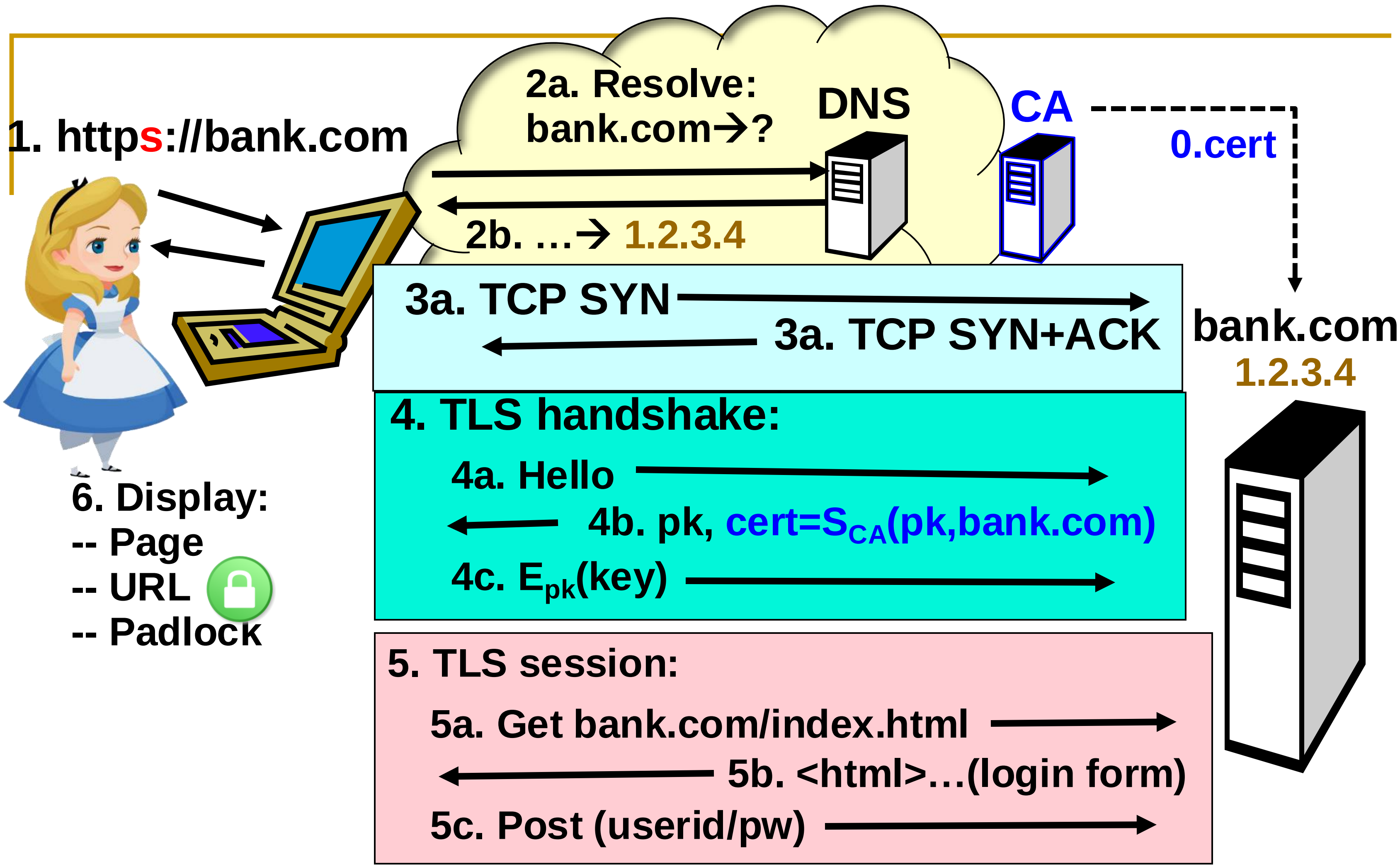
Second site…


Bank of America - Banking, Cre
bankofamerlca.com
Personal
Small Business
Wealth Management
BANK OF AMERICA
Checking
Savings & CDs

This (third) site is phishing. Can you tell? How?

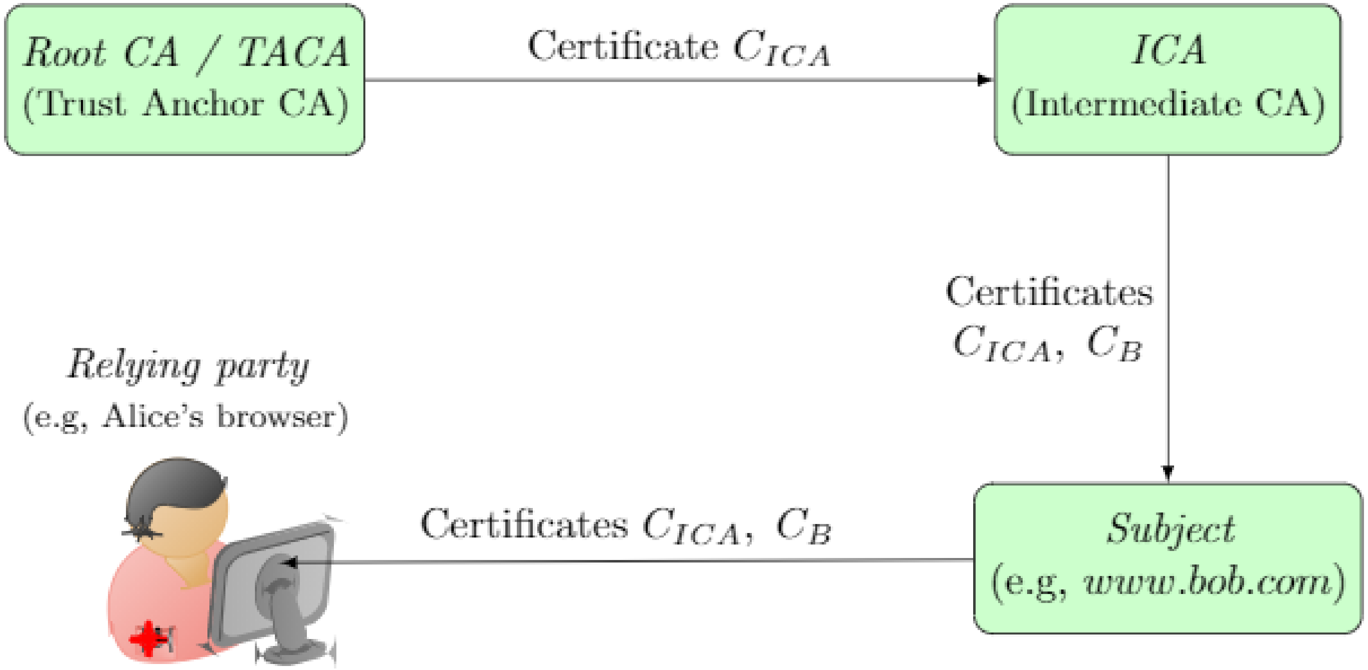


Web Security with TLS/SSL (simplified)



Web-PKI

- Browsers contain keys of **Root CAs** (trust anchors)
- Root CAs defined by **root program**
  - Of Google, MS, Mozilla, Apple
- Subject (website) certs issued by intermediate CA



Some infamous PKI failures

2001	VeriSign: attacker gets code-signing certs
2008	Thawte: email-validation (attackers' mailbox)
2008,11	Comodo not performing domain validation
2011	DigiNotar compromised, over 500 rogue certs discovered
2011	TurkTrust issued intermediate-CA certs to users
2012	Trustwave issued intermediate-CA certificate for eavesdropping
2013	ANSSI, the French Network and Information Security Agency, issued intermediate-CA certificate to MitM traffic management device
2014	India CCA / NIC compromised (and issued rogue certs)
2015	CNNIC (China) issued CA-cert to MCS (Egypt), who issued rogue certs. Google and Mozilla removed CNNIC from their root programs.
2013-17	Audio driver of Savitech install root CA in Windows
2015,17	Symantec issued unauthorized certs for over 176 domains
2019	Mozilla, Google *software* blocks *customer-installed* Kazathhstan root CA (Qaznet)
2019	Mozilla, Google revoke intermediate-CA of DarkMatter, and refuse to add them to root program

2019: Blocking Qaznet

- Kazakhstan gov't requires installation of new root CA: Qaznet
- Detected use for MitM on users
- Mozilla, Google browsers reject Qaznet CA
  - Even when installed by user !
- Kazakhstan's response ?
  - Hint: in 2020 ?

Why and How CAs fail?

- Many CAs `trusted' in browsers (as root)
- Every CA can certify any domain (name)
  - Name constraints NOT applied (esp. to roots)
  - Some CAs may be negligible or even rogue
- Limited requirements to become CA
- Often, minimal / no liability/damage after CA failures
- ➔ Rogue Cas and negligent CAs
- Can we improve defense against bad CAs?

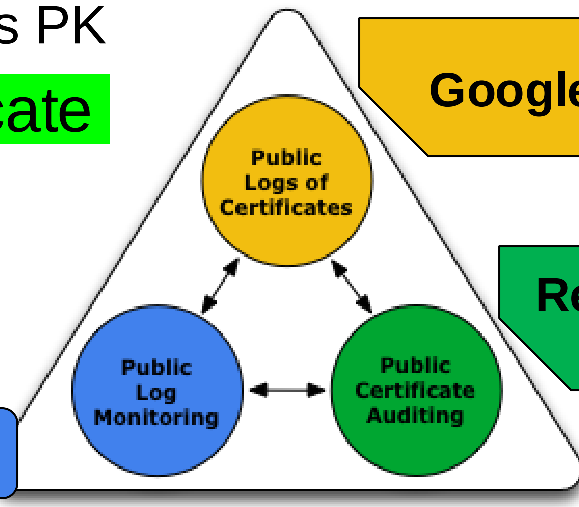
Defenses against CA failures

- **Use name constraints to limit risk**
  - who can issue global TLDs (.com, etc.)?
- **Static key pinning:** `burned-in' public keys
  - Detected MitM in Iran: rogue DigiNotar cert of Google
  - Limited: changing keys? Which keys to preload ?
- **Dynamic Pinning: HTTP Public-Key Pinning (HPKP)**
  - Server: I always use this PK / Cert / Chain
  - Client: remember, implement, detect & report attacks
  - Concerns: key loss/exposure, changing keys (recover security)
- Certificate Transparency (CT): **Accountability**
  - **Public, auditable certificates log**

Certificate Transparency (CT) [RFC6962]

- X.509, PKIX: CAs sign cert
  - **Accountability**: identify issuer, **given** (rogue) cert
- Challenge: find rogue cert
  - Unrealistic to expect relying parties to detect !
  - Google detected in Iran - since Chrome had pinned Google's PK
- Proposed solution: Certificate Transparency
- Functions: **Logging, Monitoring and Auditing**

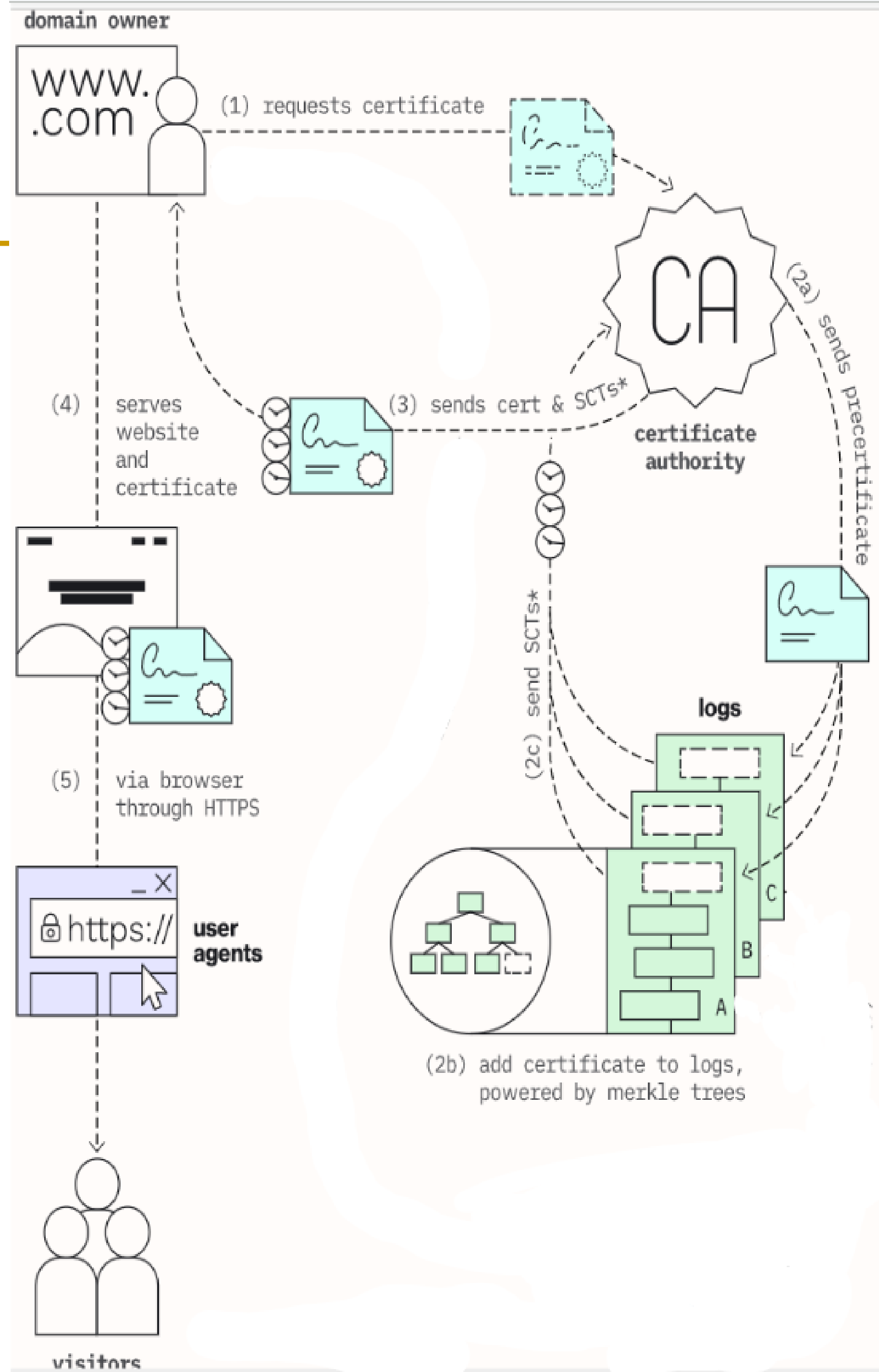
- **Loggers** provide public logs of certificates
- **Monitors** monitor certificates logged for detection of suspect certificates
- **Auditing** (auditors?): check for misbehaving loggers



Google, CAs,…

Relying party (browser)

CAs, Facebook, others

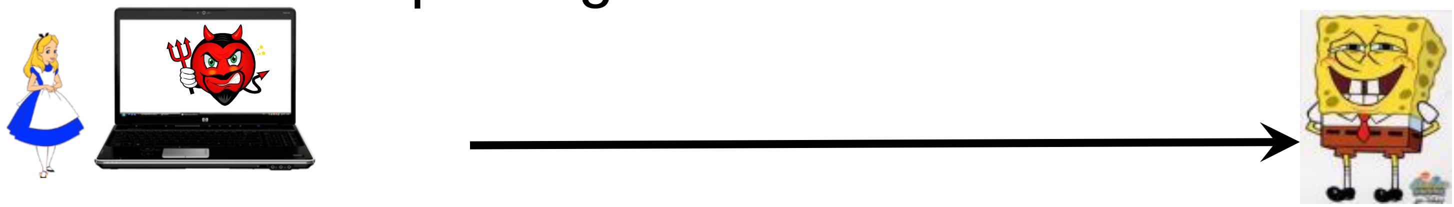

domain owner
www.
.com
(1) requests certificate
CA
(2a) sends precertificate
(3) sends cert & SCTs*
certificate
authority
(4) serves
website
and
certificate
(2c) send SCTs*
logs
(5) via browser
through HTTPS
C
B
A
https://
user
agents
(2b) add certificate to logs,
powered by merkle trees
visitors

Malware: by infection method

- From malicious website: zombie, puppet or cross-site script
  - A major goal of spoofed websites, phishing attacks

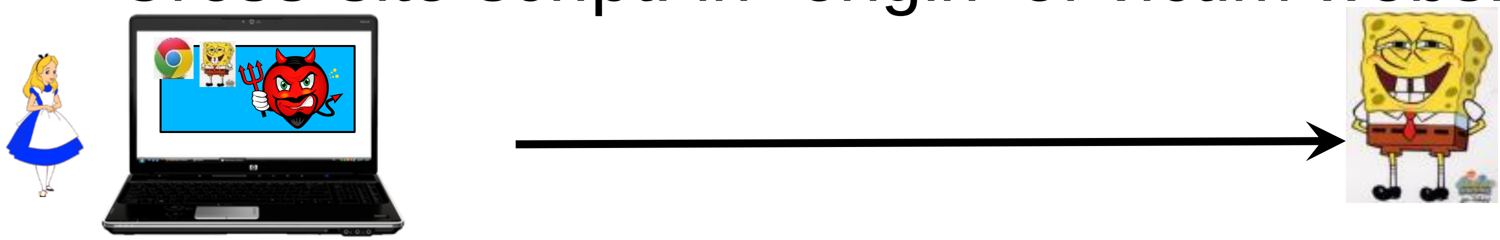
Malware on clients: by capability

- Bot/Zombie - `Man-in-the-Browser/Host'
  - Different privileges: rootkit / admin / user / extension



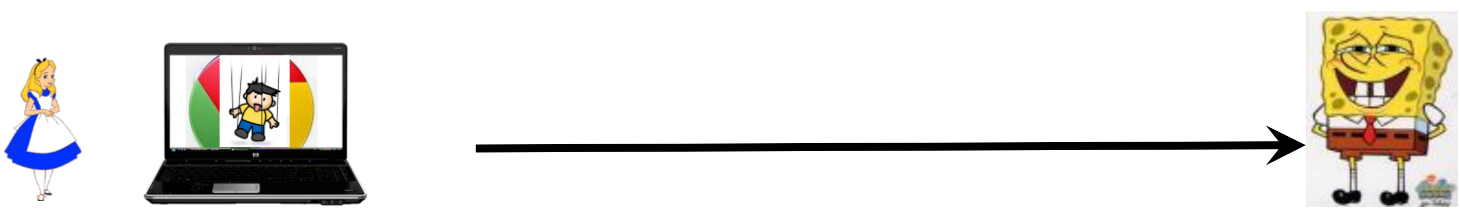
Bob.org

- Cross-site script: in `origin' of victim website



Bob.org

- Puppet - `Man-in-the-Sandbox'



Bob.org

Malware: by infection method

- From malicious website: zombie, puppet or cross-site script
- Installed by user: Trojan horse
  - HW/SW
- From other malware on host: Virus
  - Virus: malware appended to victim program
  - Executed when (infected) victim executes
  - Searches for and infects other victim programs
- From malware in other host: Worm
  - Searches for and infects other victim hosts

Malware: by goals and control

- Control, communication, updates
  - Botnet
  - Covert channel
- Goals
  - Remote control (backdoor)
  - Ransomware
  - Information, e.g., key-logger
  - Unauthorized operations: from banking malware to adware
  - Denial of Service (DoS), e.g., Time-bomb
  - Resources: storage, reputation (spam), network (DoS)…

Defenses against malware

- Blacklist: identify known malware
  - By contents/hash (easy to escape detection)
  - By behavior, e.g., sys-calls (a bit harder to escape)
- Whitelist: Use only good (signed) software
  - Who will validate it? How? Liability?
    - Reality: very limited – but still quite effective
  - Can we allow users to <u>create</u> software ???
- ‘Greylist’: allow non-whitelisted SW but…
  - Limit its capabilities to reduce risk
  - Detect malicious or ‘suspect’ behavior

Detecting known malware

- Detect 'Malware signature' (bad term!)
- Hash of malware as a signature
  - Easily evaded, e.g., polymorphic malware
    - A 'packer' creates randomized versions of malware
    - Randomized malware runs a small 'loader'
      - Unpacks the randomized malware and runs it
- Sketch/behavioral signature (e.g., system calls)
  - Harder to evade, but also to avoid false-positives
- Fail against **new** malware
  - Esp. if it can test signature!

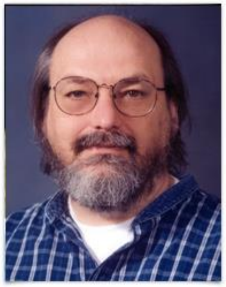
Detecting malicious/suspect behavior

- Detect malicious (or ‘suspect’) behavior
  - Can’t detect algorithmically
    - Halting Theorem [Cohen]
  - Heuristic and partial detectors; many use ML
  - Concerns: false-positive, adversarial learning
- Host (victim) detection
  - Detection may be too late…
  - And often not properly deployed, updated…
  - ‘Firewall’ detection: run in virtual machine
  - Challenges:
    - Malware detects VM
    - Malware behavior invoked only after interaction, time

Inspecting Software for Malware

- Assume ‘wizards’ **can** detect malware, **given source code**
  - Not realistic… but assume it anyway for now
  - Recall: detection is intractable computationally
  - Hard enough; surely can’t hope to find in binary!
- Idea: use only source inspected by Wizard
- Open-source **inspect-compile-use** method:
  - Wizard inspects source code S
  - Only if Ok, compile: E← C(S) , use executable E
- [Thompson84]: fail if compiler C is adversarial

Thompson's adversarial compiler (1)

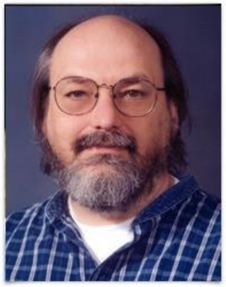


- Adversarial compiler $C1_A$: given valid source S, output executable with trapdoor $C1_A(S)$
  - E.g., for login program:

```
C1_A(S) {
    if (match(S, "login-pattern")) {
        output executable for login-with-backdoor
        return
    }
.... /* compile as usual */
  }
```

- Conclusion 1: use only compiler inspected by Wizard!
- But: wizard inspects source of compiler, not executable, so…

Thompson's adversarial compiler (2)



- Adversarial compiler C_A : given valid source S, output executable with trapdoor C_A(S) :

```
C_A(S) {
    if (match(S, "login-pattern")) {
        output executable for login-with-backdoor
        return
    }
    if (match(S, "compiler-pattern")) {
        output executable for compiler-with-backdoor
        return
    }
    .... /* compile as usual */ ……….. }
```

- Trapdoors persist in compiled-compiler, login !

So far, mostly bad news…

- In practice:
  - Unintentional flaws are unavoidable
  - Flaws, and vulnerabilities, are often hard to detect (even in source code)
  - Trapdoors (intentional flaws) can be very hard to detect (even in source code)
- In theory:
  - Detecting flaws/trapdoors is intractable
  - Even if wizard could detect all flaws/trapdoors in source code, rogue compiler can put trapdoor in executable
- So, any hope to ensure benign executable??
  - Suppose we trust some (old?) compiler C_T …
  - But want to use another (better?) compiler C_A [why?]

[Wheeler'05]

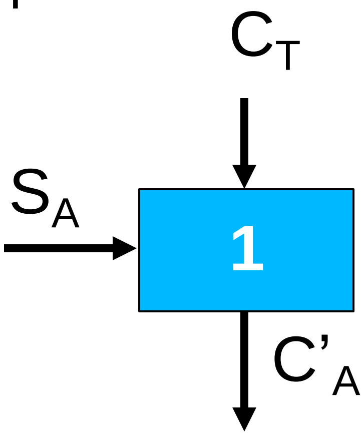
Diverse Double-Compiling (DDC)

- Assume: C_T : executable of trusted compiler, and:
- Source S_A and exe C_A of another (untrusted) compiler
  - Source code S_A **validated by wizard**
  - In typical case, should hold: $\mathbf{C_A=C_A(S_A)}$
- How can we use C_T to validate C_A ?

[Wheeler'05]

Diverse Double-Compiling (DDC)

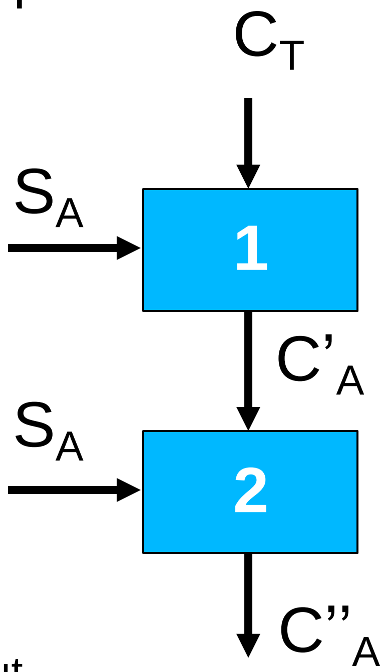
- Assume: C_T : executable of trusted compiler, and:
- Source S_A and exe C_A of another (untrusted) compiler
  - Source code S_A **validated by wizard**
  - In typical case, should hold: $\mathbf{C_A=C_A(S_A)}$
- How can we use C_T to validate C_A ?
- Idea 1: use C_T to compile S_A , i.e., $C'_A=C_T(S_A)$
- Now what?
  - Idea 1.1: use C'_A instead of using C_A (same source code!)
  - But: C_A may be better (optimized)!
  - Idea 1.2: confirm that $C_A=C'_A$
  - But: this would often fail even for benign C_A (e.g., if it optimizes)
  - So Wheeler found another (better?) idea…



[Wheeler'05]

Diverse Double-Compiling (DDC)

- Assume: C_T : executable of trusted compiler, and:
- Source S_A and exe C_A of another (untrusted) compiler
  - Source code S_A **validated by wizard**
  - In typical case, should hold: $\mathbf{C_A=C_A(S_A)}$
- How can we use C_T to validate C_A ?
- Idea 1: use C_T to compile S_A , i.e., $C'_A=C_T(S_A)$
- Idea 2: use C'_A to compile S_A (again!), i.e., $C''_A=C_T(S_A)$
- Now what??
  - Confirm that $\mathbf{C_A=C''_A}$!!
  - Should be true, since C'_A may be less efficient than C_A but should have the same functionality as C_A
  - Assuming…
    - C_A is deterministic, stateless, time-invariant, and compiled using itself: $\mathbf{C_A=C_A(S_A)}$
    - Wheeler confirmed this works for few (typical) compilers; **extend for cross-compiler!**



Summary: Vulnerabilities, Malware & Ethics

- Networks ➔ complexity ➔ vulnerabilities
  - Keep it Simple (KISS) principle
  - Industry focuses on product vulnerabilities;
    we focus on protocol and config vulnerabilities
- Malware: another major threat
  - Validation of software is impractical
  - Validating compilers? Even less practical
- Challenging ethical dilemmas
  - E.g., dual-use pen-testing and reconnaissance
- **Knowledge is Power !!**