Application Security (apsi)

Lecture at FHNW

Lecture 5, 2020

Arno Wagner, Michael Schläpfer, Rolf Wagner

<arno@wagner.name>, <{michael.schlaepfer,rolf-wagner}@fort-it.ch>

Agenda

- Security Testing
- Backdoors in Software
- Economic Aspects of Security

Security Testing (for Software)

- Pen-Test: Attempt to break in
  - Long catalog of things to try…, for example Fuzzing, injection, default credentials...
- (Load-test: Determine high-load behavior)
- Code Inspection and review
- Code scanners (example: Fortify)
- Code emulation environments (example: Valgrind or Qemu)

Pen-Test

There are 3 (main) classes:

1) White Box: Tester has credentials (passwords), documentation, maybe even debug access
2) Black Box: Tester is given minimal information (target IP range) not more
3) Grey Box: Somewhere between White Box and Black Box

- In actual reality, Black Box is least useful, except as exposure-test
- Typical situation is Grey Box, often because no full information is available
- Sometimes Pen-Test may only be run against test-environments
- Often, Pen-Tests may not do flooding and must be done in off-hours
- Pen-Tests can always break the target system, no matter how carefully done
- Interesting reading: https://threader.app/thread/1063423110513418240

How does a Pen-Test fail?

It fails to successfully attack the system!

What do you know in that case?

- Nothing!
  → Customer feels secure, but tester may just have tested the wrong things
  → Risk is highest in a Black Box test

The customer just fixes the observed issues, nothing else

- Pen-Tests can sometimes identify root-causes – these need to be fixed
  → Example: Unpatched software indicates broken software maintenance
- Pen-Tests are not complete tests (also due to budget-restrictions)
- Pen-Tests are useful to create awareness

Code Inspection and Review

A second person looks at the code and looks for problems

- Limited usefulness if done internally
- Critically dependent on reviewer skill and available time

Direct results (somewhat useful):

- Bugs

Indirect results (very useful, but politically problematic):

- General code quality
- Interface quality
- Skill-level of original coder
- ...

Code Scanners

A code scanner is a tool that looks for problematic code

- Can be simple (structural) pattern matching
  Example finding "if (a=b) {}" and the like
- Can be very sophisticated
  - Data-flow techniques (similar to taint-checking)
  - Check whether input values were looked at at all
  - Memory-leak candidates
  - …
- Not too easy to use (depends) and may create false sense of security
- May collide with coder ego….

Execution Emulators

Example 1: Valgrind

- Executes code in symbolic form
- Uses JIT and other optimization techniques
- Most useful as memory-debugger (overflows, memory-leakage, …)
- Around 20-25% of original execution speed

Example 2: Qemu

- Software virtualization tool
- Extended debugging options
- Allows non-native configurations (different CPU, etc.)
- Pretty slow...

Code Analysis for Security

Aim:

- Find vulnerabilities
- Estimate overall code quality

These days often: Finding of intentionally placed backdoors

  - In code
  - In libraries, run-time environments, containers, VM images machines, virtualization software,…
  - In cryptography
    → This is impossible for modern "NOBUS" backdoors, but the possibility ("compromised design"/"compromised algorithm") may be identifiable

- Placed by:
  - Disgruntled employees (defense: keep your employees happy…)
  - State-sponsored sabotage
  - Vendors that want marketing data without asking/telling the user

Backdoors in Code

Definition

- A remotely or locally accessible (hidden) undocumented functionality (often a hidden interface) that gives an attacker that knows about it more access than the owner of the software is aware or has authorized.

  Note: This does include bypassing data-access restrictions

Additional characteristics

- Uses network camouflage
  Example: port open only at specific times or after specific events, e.g. after port-knocking (connection attempts to a sequence of specific non-open ports) or "login-knocking" (failed logins with specific timing or for specific users)
- May be disguised as coding mistake
- Intentionally made hard to detect on code-level

Backdoor Hiding Techniques

- Meta-technique: Use what appears to be common coding errors
  → If you find it, you do not know whether this was an attack!
  (Remember that coders can be arbitrarily incompetent these days...)
- Use debug code that was "accidentally" left active
- Omit workarounds for known vulnerabilities or implement them only partially
- Use violations of "least surprise" in libraries and system calls or services
  - Use obscure and complex library functionality
- Use race-conditions
- Use bad initialization
- Use intentionally bad or misleading code comments
  → Works well for interface specifications of complex functions
- Use Intentionally obfuscated, complex and/or badly structured code
- Use low-skill coders and then look for vulnerabilities in their code

Finding Backdoors

Using tools (Security scanners):

- Only work if the attacker has not tested against them (many do...)
- Only work if code does not give lots of errors ("hiding a tree in the forest...")

Manually:

- Identify all input from outside and follow the respective data-paths
- Look for functionality that "does not make sense", like very awkward code, unnecessary complex code, complex libraries that are not really used, etc…
- Look for misleading comments and comments that do not make sense
- Verify <u>all</u> functionality

If done right, this is generally more expensive than rewriting the software!

- (Partial) code rewriting with trusted personnel is a valid approach
  → May still have cost advantage if design/architecture is reused.

Backdoors in Cryptography

Relatively new trend: "NOBUS" backdoors

- NOBUS = "NObody But US"
- Uses cryptographic properties to protect the backdoor
- Is not distinguishable from secure version without a secret key

=> Only the attacker can see them

This is a "mathematically compromised design"

How do deal with it?

- Assume if the possibility is there, then it is used!
- The secret protecting the backdoor may leak.
  => <u>Do not ever trust these systems!</u>

Example 1: ECC

ECC (Elliptic Curve Cryptography) relies on a selected curve

- Generation of a curve can be done in a way to include a backdoor
  → Attacker generates curve, publishes it
  → Attacker has secret knowledge of curve property that facilitates attack

Defense:

- Use only curves that are generated by an "obviously" not compromised procedure and generated by a trusted party
  → Example: Curve25519 Reference: https://safecurves.cr.yp.to/

Example: Dual_EC_DRBG

- Uses a curve that "fell from the sky" (Well, the sky over Fort Meade...)
- Demonstrated to be vulnerable with other, specifically generated curve

Advice: If in doubt, do not use ECC at this time

Example 2: Specially selected Primes for Discreet Logarithms

Idea:

- Select a Prime P so that the discrete logarithm over the generated finite Field is easy to compute
- Detecting this "trapdoor" if P is unknown is likely computationally infeasible

Protection:

- Use your own prime(s)
  → Use methods where each party generates their own primes

Reference: http://eprint.iacr.org/2016/961

Economic Aspects of Security

Scope:

- Commercial software (COTS)
- Custom-software (self-built or built-to-order)
- Not in scope: FOSS (that one is more difficult) unless commercially used

Key-Question: Is there profit in insecure software?

- Yes, there is!

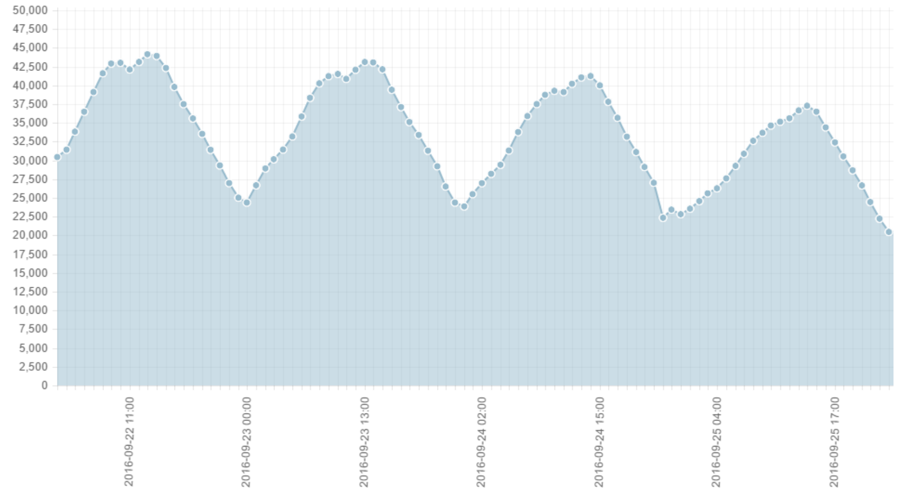
Case Study: DDoS on Brian Krebs (20.9.2016)

The observed problem:

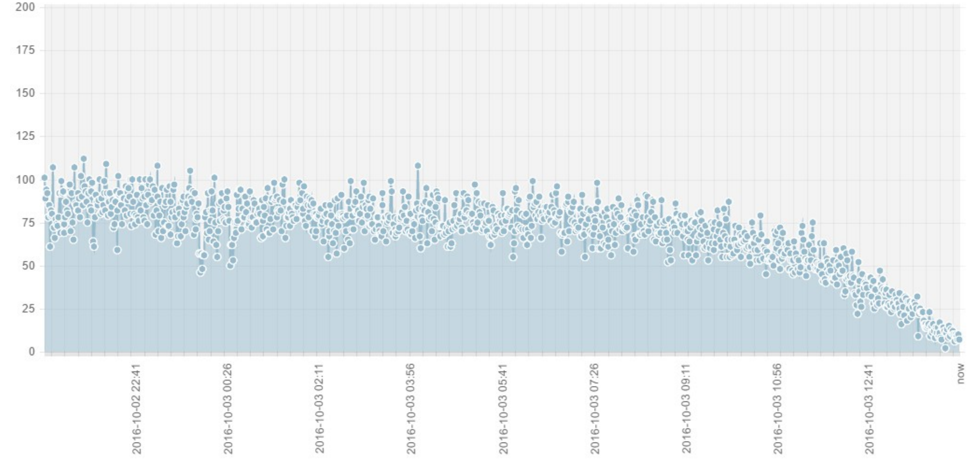
- Simple flooding with 622Gbit/s
- Akamai stopped its (free) DDoS-protection after a few hours due to cost
  - Google took over a few days later (also free)
- Suspected attackers are a "DDoS as-a-service" company where Krebs got the owners arrested
- Not very smart: Criminal "best-practice" is to only be an annoyance
  → 622Gbit/sec is a real danger ("small-time criminals with nukes")
- Endangers the bot-net used

What can Perform Such an Attack?

Regular DoS:



This one:



Source: https://www.malwaretech.com/2016/10/mapping-mirai-a-botnet-case-study.html

IoT: Internet of insecure Things

Here: Mostly ElCheapo Internet-connected surveillance cameras

- Run without firewall isolation by most users
- May open firewall by using UPnP
- No good management software
- Default password and user
- Even if pwd/user changed for web-interface: Telnet still at default
- No upgrade or patching path

This does not even require any real attack!

An attacker estimated he could compromise 380 million machines/day

Why are These IoT Devices so Insecure?

What would have helped?

- Per-device passwords and usernames
- Password-setting done right (change all of them)
- Automated patching?

But:

- Requires some (minimal) understanding of security
  - More expensive employees or external expertise needed
- Makes devices minimally more expensive (less/no profit)
- May create support-cost

And nothing bad happens to the manufacturer as things are set up now!

Possible Solutions

- Require a "Security Quality" certification
  → These are often pretty useless and are mainly used to protect markets
- Block insecure devices via customs
  → See above
- Require vendors to do recalls
  → And what if they do not care? Or close the company and open a new one?
- Require ISPs to block problematic devices
  → Unclear how to identify them
  → Infrastructure does not really exist for that
  → May hit a lot of innocents
- Make the user/owner liable
  → That will go over well...
- Raise awareness
  → They do not seem to care at this time already….
- ???

Does it Pay to make Software Secure?

Question: Cost of attack * frequency vs. cost of making software secure

- Unfortunately, being insecure (seems) often cheaper
- Attacks are not a major cost-factor: https://www.schneier.com/blog/archives/2016/09/the_cost_of_cyb.html
  - Survey over 12,000 incident reports 2004-2014
  - Cost per attack: Average $200'000 which is only about 0.4% of annual revenue
    For comparison: Fraud is around 6% of annual revenue
    → Not seen as a major issue
- Preventing attacks is expensive

Sometimes this calculation fails (but mostly people seem not to care...)

- Example: The offer for Yahoo shrunk from 4.8 billion to 3.8 billion once their >1Million customer records breach became known.

We are missing "Reference-Catastrophes" that normal people understand

So, what to do?

This cannot go on for much longer (or can it?)

- Implement working product liability and require insurance
  - May work in certain markets...
  - …but what standards to apply?
- Require certifications
  - Well, see above. These are often not worth much.
- Require independent reviews
  - Helps to some degree, but these are expensive, so people try to do without
- Improve CS education to teach IT Security as mandatory topic
  - Not generally done, even today
- Make the coders liable for insecure code? ("Malpractice"?)
  - We do not even have standards who is allowed to write critical code!

→ Expect this to be a topic that will grow more important for a long time