OBSFUSCATION

THE HIDING OF INTENDED MEANING, MAKING COMMUNICATION CONFUSING, WILFULLY AMBIGUOUS, AND HARDER TO INTERPRET.

CS6410

Ken Birman

Spafford's Concern

- Widely circulated memo by Gene Spafford:
  - Consolidation and virtualization are clearly winning on cost grounds, hence we can anticipate a future of extensive heavy virtualization
  - Xen-style sharing of resources implies that in this future the O/S will have privilaged components that can peek across protection boundaries and hence see the states of all hosted VMs without the VMs realizing it
  - Could leak information in ways that are undetectable
- ... but it gets worse

Spafford's Concern

- … and what if a virus breaks out?
  - In a standard networked environment, viral threats encounter some substantial amount of platform diversity
  - Not every platform is at the identical "patch" level
- In a data center with virtualization, every node will be running identical versions of everything
  - At least this seems plausible, and this is how well-managed enterprises prefer things
  - The resulting "monoculture" will be fertile ground for a flash-virus outbreak

Could this risk be real?

- Clearly, computing systems are playing socially critical roles in a vast diversity of contexts
  - Are computing platforms, networks and the power grid the three “most critical” infrastructures?
    - The case isn’t hard to make…
    - … although it is confusing to realize that the are mutually interdependent!
  - Massive outages really could cause very serious harm
- On the other hand, is Spaf’s scenario really plausible, or is it just a kind of empty fretting?

… dire conclusions

- Within a few years, consolidated computing systems will be the only viable choice for large settings
- Government systems will inevitably migrate to these models under cost pressure, but also because there will be nothing else to buy: the market relentlessly reshapes itself under commercial pressure
- And so everything – literally everything – will be vulnerable to viruses.
- The world will end.

Chicken Little has a long history in CS…

- Children's tale should be a caution

- Not every worry is plausible
  - Those who specialize in worrying think in larger terms, because there are too many things to worry about!
  - Real issues are dual:
    - How big is the (likelihood * damage) "estimate"?
    - And how likely is this, in absolute terms?
    - Below some threshold we seem to ignore even end-of-world scenarios. Basically, humanity believes bullets can be dodged…

… so how should we view monocultures?

- Fred Schneider and Ken were asked to run a study of this by the government; we did so a few years ago.
  - How does one study a catastrophy prediction?
  - What does one then do with the findings?

- We assembled a blue-ribbon team in DC to discuss the issues and make recommendations

Composing the team

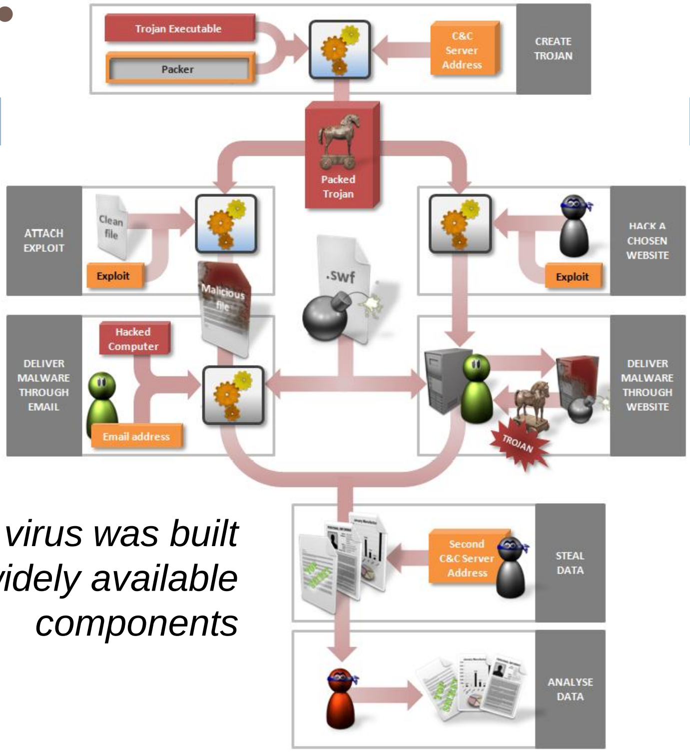
- We picked a group of people known for sober thinking and broad knowledge of the field
  - This include NSA “break in” specialists
  - Industry security leaders, like Microsoft’s top security person
  - Academic researchers specializing in how systems fail, how one breaks into systems, and how to harden them
  - Government leaders familier with trends and economic pressures/considerations

A day that split into three topics

- Question one: Is there a definable problem here?
  - … that is, is there some sense in which consolidation is clearly worse than what we do now?
- Question two: How likely and how large is it?
- Question three: What should we recommend that they do about it?

Breaking into systems…

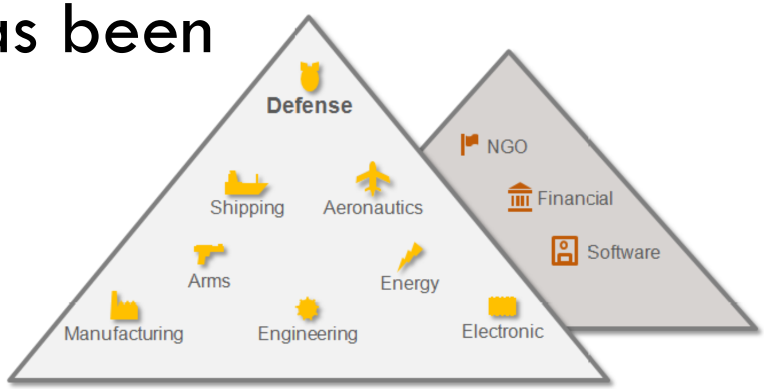
- … is, unfortunately, easy



The virus was built using widely available components

- Sophisticated "rootkit" products help the attacker

- Example: Elderwood Gang has been very successful in attacking Adobe & Microsoft platforms



Elderwood Targets

Elderwood… one of <u>thousands</u>

- The numbers and diversity of viruses is huge, and rapidly increasing

- NSA helped us understand why
  - Modern platforms of all kinds are "wide open"
    - O/S bugs and oversights
    - Even wider range of <u>*application*</u> exposures
    - Misconfiguration, open administrative passwords, etc
  - Modern software engineering simply can't give us completely correct solutions. At best, systems are robust when used in the manner the testing team stress-tested.

Could we plug all the holes?

- NSA perspective:
  - A town where everyone keeps their jewelry in bowls on the kitchen table...
  - ... and leaves the doors unlocked
  - ... and the windows too
  - ... and where the walls are very thin, in any case
  - ... not to mention that such locks as we have often have the keys left in them!

Expert statistics

- Virus writers aim for low-hanging fruit, like everyone else
  - Why climb to the second floor and cut through the wall if you can just walk in the front door?
  - Hence most viruses use nearly trivial exploits

- This leads to a “demographic” perspective on security: if we look at “probability of intrusion” times “category of intrusion”, what jumps out?

Configuration exploits!

- By far the easiest way to break in is to just use a wide-open door into some component of the system, or application on the system

- These are common today and often are as simple as poor configuration settings, factory passwords, other kinds of "features" the user was unaware of
  - For example, some routers can clone traffic
  - And many routers have factory-installed web accessible pages that allow you to access their control panel
  - Hence if the user has such a router you can clone all their network traffic without really "breaking in" at all!

Configuration exploits

- Another very big class of configuration issues are associated with old and insecure modes of operation that have yet to be fully disabled
  - Many old systems had backdoors
  - Some just had terrible ad-hoc protection mechanisms

- When we install and use this kind of legacy software we bring those exposures in the door
  - Even if we could easily “fix” a problem by disabling some feature, the simple action of doing that demands a kind of specialized knowledge of threats that few of us possess

Broad conclusion?

- Computers are often loaded with "day zero" vulnerabilities:
  - The attack exploits some kind of a feature or problem that was present in your computer the day it arrived
  - Vendor either didn't know about it or did know, but hasn't actually fixed it
  - Your machine is thus vulnerable from the instant you start using it.
- Sometimes also used to describe an attack that uses a previously unknown mode of compromise: the vulnerability becomes known even as the attack occurs

Good platform management

- An antidote to many (not all) of these issues
  - Highly professional staff trained to configure systems properly can set them up in a *much* more robust manner

- Best practice?
  - Experts examine every single program and support a small, fixed set of combinations of programs, configured in ways that are optimal
  - Every machine has the right set of patches
  - End-users can't install their own mix of applications, must chose a configuration from a menu of safe choices

Synthetic diversity

- Obsfuscation goes one step further
  - Start with one program but generate many versions
  - Use compiler techniques or other program transformation (or runtime transformation) tools to close attack channels by making platforms “systematically” diverse in an automated manner
  - Idea: if an attacker or virus tries to break in, it will confront surprises because even our very uniform, standard configurations will exist in many forms!

Historical perspective

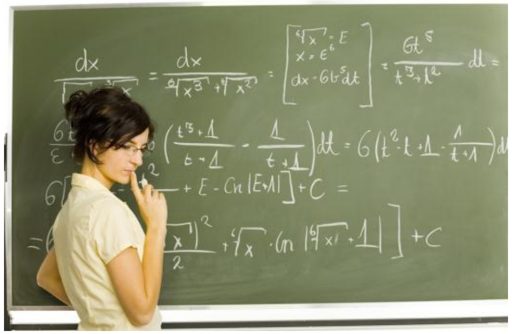
- Earliest uses focused on asking developer teams to create multiple implementations of key elements
  - Idea was to get them to somehow "vote" on the right action
    - Puzzle: Suppose A and B agree but C disagrees
    - Should we take some action to "fix" C? What if C is correct?
  - Nancy Levinson pointed out that specification is key: a confusing or wrong specification can lead to systematic mistakes even with a diverse team
    - Also found that "hard parts" of programs are often prone to systematic errors even with several implementations
  - Still, technique definitely has value

French TGV brake system

- TGV is at grave risk if brakes fail to engage when there is danger on the track ahead
- How to build a really safe solution?
  - One idea: flip the rule. Brakes engage unless we have a "proof" that the track ahead is clear
  - This proof comes from concrete evidence and is drawn from a diversity of reporting sources
  - But we also need to tolerate errors: weather and maintanence can certainly cause some failures

So are we done?

- French engineers pushed further
  - Rather than trust software, they decided to prove the software correct using formal tools
  - Employed formal methods to specify the algorithm and to prove their solution correct.
  - Then coded in a high level language and used model-checking to verify all reachable states for their control logic



- *But what if the proof is correct but the compilation process or the processor hardware is faulty?*

Next step

- They generated multiple versions of the correct algorithm using a grab-bag of heuristics to transform the code "systematically"

- Now the original single program exists as a set of k variant forms that vote among themselves
  - The brake hardware itself implements the voting in a mechanical way
  - Two votes out of three wins... space shuttle used the same idea to protect the cargo door latches

Other uses of this idea?

- Ronitt Rubenfeld used it to overcome bugs in the implementation of complex numerical functions
  - She looked at continuous mathematical functions with complicated implementations
  - Rather than compute F(x), she would compute a series of values: $F(x-2\delta)$, $F(x-\delta)$, $F(x)$, $F(x+\delta)$, $F(x+2\delta)$
  - Then used the continuity of the underlying function to compensate for possible mistakes at "isolated" points
- The technique works well and can overcome bugs known to have been present in released math libraries that entered widespread use

Applying diversity to programs

- Various options present themselves
  - We can "permute" the program code in ways that preserve behavior but make the layout of the code hard to anticipate
  - We can pad heap-allocated objects with random "junk"
  - We could replace one variable with a set of replicas
  - We could vary the location of the heap itself
  - We could renumber the O/S system calls on a per-platform manner
  - Use different versions of the system-call library for each application we build...

Synthetic Diversity really works!

- With aggressive use of these techniques our data center ends up with thousands of non-identical clones of the “identical” platform!
  - Each one differs in details but has same functionality
  - Virus is very likely to be confused if it tries to exploit array-bound overrun or similar bugs: Attack becomes a non-event, or causes a crash
  - Much evidence that these techniques genuinely eliminate much of that low-hanging fruit

What problems persist?

- Functionality attacks will still be successful
  - Example: SQL code injection attack: on a web form that asks, e.g., for a name, provide “code”
  - Consider this query:

    From a form or RPC argument

    - statement = "SELECT * FROM users WHERE name = '" + **userName** + "';“
  - Now set “userName” to ' or '1'='1
    - SELECT * FROM users WHERE name = ‘’ or 1=1

Synthetic diversity limitations

- Can’t protect against “legitimate” APIs used in unintended ways, or that can be combined with undesired consequences

- Can’t help if attacker has a way to discover a password or can manipulate network traffic or has a trick to “snapshot” your address space at runtime
  - Not too hard to find sensitive content even if it moves from place to place

What comes next?

- Would it be feasible to *compute on encrypted data?*
  - *(Without decrypting it first)*
- Many modern platforms include a hardware TPM: a form of trusted chip with secret keys baked in
  - Chip can do cryptographic ops: encrypt, decrypt, sign
  - But can't be tricked into disclosing the key itself
- Suppose we could somehow leverage this to compute on data while in encrypted form

What comes next?

- Better O/S architecture with stronger built-in protection features
  - Modern O/S is far too permissive
  - Trusts things like the file system data structure, willing to "mount" a disk without checking it first
  - We install new applications and shell or browser extensions all the time!
- Perhaps a stronger O/S security model could help
- But on the other hand, would market accept it?

Hidden hand of the market

- The ultimate decision tends to be market-driven
  - Trend in favor of cloud and virtualization/consolidation is a market (economics)-driven phenomenon
  - Money goes to the cheapest, most scalable story in town and eventually, the expensive “better” option fails
- How have markets viewed diversity mechanisms?
  - By and large, they reject these solutions!
  - Even the ones that are “transparent” to users

Issue: Nothing is really transparent

- Imagine a program with a serious heisenbug that is masked by some fluke of the runtime setup or compiled code layout
  - E.g. it sometimes reads past the end of of a string, but by accident, the next object is always zero and hence this terminates the string
  - Suppose that this passes Q/A and doesn't crash
- Now apply synthetic diversity tool...
  - ... that "working" application starts to segment fault!

Large production users fear such issues

- If Oracle starts to crash on my platform, I have few options to fix the issue
  - Debugging the Oracle source-code is not one of them
  - Paying for an urgent fix might break my budget
  - Disabling the synthetic diversity tool could be the best option available

- Many platform owners have reasoned this way
  - After all, even with diversity, all we've done is to close the front door and perhaps removed the key from the lock

Conclusions

- Modern systems are really wide open to attack
- Consolidation onto the cloud or other virtualized platforms could benefit in many ways
  - Standard, professional administration could close configuration problems and ensure proper patch level
  - At least zero-day issues will mostly be removed
- Diversity can take us further
  - Won't solve the real issue, but can really help