Detecting Attacks, cont.

CS 161: Computer Security

Prof. David Wagner

April 8, 2016

Special request: Please spread out!
Pair up. Each pair, sit far away from anyone else.
If you're just arriving, sit next to someone who is alone.

Specification-Based Detection

- Idea: don't learn what's normal; specify what's allowed
- FooCorp example: decide that all URL parameters sent to foocorp.com servers **must** have at most one '/' in them
  - Flag any arriving param with > 1 slash as an attack
- What's nice about this approach?
  - Can detect novel attacks
  - Can have low false positives
    - If FooCorp audits its web pages to make sure they comply
- What's problematic about this approach?
  - Expensive: lots of labor to derive specifications
    - And keep them up to date as things change ("churn")

Styles of Detection: Behavioral

- Idea: don't look for attacks, look for evidence of compromise

- FooCorp example: inspect all output web traffic for any lines that match a passwd file
- Example for monitoring user shell keystrokes:

  `unset HISTFILE`
- Example for catching code injection: look at sequences of system calls, flag any that prior analysis of a given program shows it can't generate
  - E.g., observe process executing **`read`**(), **`open`**(), **`write`**(), **`fork`**(), **`exec`**()   …
  - … but there's *no code path* in the (original) program that calls those in exactly that order!

Behavioral-Based Detection

- What's nice about this approach?
  - Can detect a wide range of novel attacks
  - Can have low false positives
    - Depending on degree to which behavior is distinctive
    - E.g., for system call profiling: **no false positives**!
  - Can be cheap to implement
    - E.g., system call profiling can be mechanized
- What's problematic about this approach?
  - Post facto detection: discovers that you definitely have a problem, w/ no opportunity to prevent it
  - Brittle: for some behaviors, attacker can maybe avoid it
    - Easy enough to not type "`unset HISTFILE`"
    - How could they evade system call profiling?
      - **Mimicry**: adapt injected code to comply w/ allowed call sequences

Inside a Modern HIDS (“AV”)

- URL/Web access blocking:
  - Prevent users from going to known bad locations
- Protocol scanning of network traffic (esp. HTTP)
  - Detect & block known attacks
  - Detect & block known malware communication
- Payload scanning
  - Detect & block known malware
- (Auto-update of signatures for these)
- Cloud queries regarding reputation
  - Who else has run this executable and with what results?
  - What’s known about the remote host / domain / URL?

Inside a Modern Antivirus

- *Sandbox execution*
  - Run selected executables in constrained/monitored environment
  - Analyze:
    - System calls
    - Changes to files / registry
    - Self-modifying code (*polymorphism/metamorphism*)
- File scanning
  - Look for malware that installs itself on disk
- Memory scanning
  - Look for malware that never appears on disk
- Runtime analysis
  - Apply heuristics/signatures to execution behavior

Summary of Evasion Issues

- Evasions arise from uncertainty/ambiguity (or incompleteness/inconsistency) because detector must infer behavior/processing it can't directly observe
  - A general problem any time detection separate from potential target
- One general strategy: impose canonical form ("*normalize*")
  - E.g., rewrite URLs to expand/remove hex escapes
  - E.g., enforce blog comments to only have certain HTML tags
- (Another strategy: analyze all possible interpretations rather than assuming one
  - E.g., analyze raw URL, hex-escaped URL, doubly-escaped URL …)
- Another strategy: fix the basic observation problem
  - E.g., monitor directly at end systems

Key Concepts for Detection

- Signature-based vs anomaly detection (blacklisting vs whitelisting)
- Evasion attacks
- Evaluation metrics: False positive rate, false negative rate
- Base rate problem

Securing DNS: DNSSEC

CS 161: Computer Security

Prof. David Wagner

Special request: Please spread out!
Pair up. Each pair, sit far away from anyone else.
If you’re just arriving, sit next to someone who is alone.

Securing DNS Lookups

- Topic for today:
  How can we ensure that when clients look up names with DNS, they can trust the answers they receive?
- But first, a diversion…

Active learning

- Today: Active learning + peer instruction
  - I’m going to ask you to work out how to secure DNS, on your own.
  - I’ll give you a series of problems. I want you to break into groups of two, decide what you think a solution might be, then report back to the class.
  - I will circulate. Ask me for help!
  - Research suggests this might be more effective than lecturing. Let’s give it a try!
- I welcome your feedback on whether it helps you learn.

Outsourcing Data Lookups

- **Problem 1.** Berkeley has a database of all its alumni, $D = \{d_1, d_2, \ldots, d_n\}$, replicated across many mirror sites. Given a name x, any client should be able to query any mirror and learn whether $x \in D$. We don't trust the mirrors, so if answer to query is "yes" (i.e., if $x \in D$), client should receive a proof that it can verify. Don't worry about proofs if answer is "no". Make performance as good as possible.

Solutions

Give to the mirror:

- Sign(Dave), Sign(Eve), ..

- To answer a query like “Dave”, response = Sign(Dave)

Solutions

Give to the mirror:

- Signatures: d1,Sign(d1),…,dn,Sign(dn)

Outsourcing Data Lookups

- **Question 2.** Suppose we use your solution, with client connecting to mirror via HTTP – but there is a man-in-the-middle (on-path attacker). What can attacker do, without being detected?

  A. Can spoof both "yes" ($x \in D$) and "no" ($x \notin D$) responses.
  B. Can spoof "yes", but can't spoof "no".
  C. Can spoof "no", but can't spoof "yes".
  D. Can't spoof either kind of response.

Authenticating “Yes” and “No”

- **Problem 3.** Same as Problem 1, except now, if answer is “no” (i.e., $x \notin D$), client should receive a proof that it can verify.

Authenticating "Yes" and "No"

- **Problem 3.** Same as Problem 1, except now, if answer is "no" (i.e., $x \notin D$), client should receive a proof that it can verify.

  Hint: Organize the data in some CS 61B data structure, then….

Authenticating “Yes” and “No”

- **Problem 3.** Same as Problem 1, except now, if answer is “no” (i.e., $x \notin D$), client should receive a proof that it can verify.

  Hint: Organize the elements as a binary tree or hash table, then….

Solutions

Say D = {Alice, Bob, Jim, Xavier}.
Give to mirror:

- Sign(C, “no”), Sign(D, no), Sign(E, no), .., Sign(Aa, no), Sign(Ab, no), Sign(Ac, no)
- Hashtable, plus Sign(i || contents of bucket i) for each I
- Sign(first, Alice), Sign(Alice, Bob), Sign(Bob, Jim), Sign(Jim, Xavier), Sign(Xavier, last)

To answer query “Doug”:

Solutions

Say D = {Alice, Bob, Jim, Xavier}.
Give to mirror:

- Sign(1, Alice), Sign(2, Bob), Sign(3, Jim), Sign(4, Xavier)
- Sign(Alice,Bob), Sign(Bob, Jim), Sign(Jim,Xavier)

To answer query “Doug”:

- Doug -> no, Bob, Jim, Sign(2, Bob), Sign(3, Jim); or Doug -> no, Sign(Bob, Jim)

Side note: CS 61B again…

If there is a data structure that can answer queries in time T(n), then there is a way to cache the data structure and have cahces provide proofs of size O(T(n)).

Why?

DNS

- **Problem 4.** Now Berkeley wants to protect its DNS records; how could it do it? What would be the advantages and disadvantages of your solution?

DNSSEC

- Guess what – you just invented DNSSEC!

- Sign all DNS records. Signatures let you verify answer to DNS query, without having to trust the network or resolvers involved.