Enhancing Byte-Level Network Intrusion Detection Signatures with Context

Robin Sommer
`sommer@in.tum.de`
Technische Universität München
Germany

Vern Paxson
`vern@icir.org`
International Computer Science Institute
Lawrence Berkeley National Laboratory
Berkeley, CA, USA

Overview

- Approaches to Network Intrusion Detection.
- Contextual Signatures.
- Example applications.
- Evaluation.
- Summary.

Approaches to Network Intrusion Detection

- Detect attacks by observing network traffic.
- Anomaly detection
  - Derive some “normal” behaviour.
  - Watch for deviations.
- Specification-based detection.
  - Define legitimate traffic.
  - Watch for violations.
- Misuse detection
  - Build a library of attack characteristics.
  - Try to detect them in the traffic.

Misuse-Detection by Signature Matching

- Signature: Characteristic sequence of bytes.
- Used by most operationally deployed NIDs (e.g. Snort).
- Advantages
  - Easy to comprehend, accumulate and share.
  - For some attacks, very tight $\longrightarrow$ Precise detection.
- Disadvantages
  - For many attacks, not tight $\longrightarrow$ False positives.
  - But if too tight, miss even tiny variations $\longrightarrow$ False negatives.
  - No notion of relevance: Was the attack successful?
- Problem: Just looking at bytes ignores (available) context.

Contextual Signatures (1)

- Goals
  - Reduce false-positives.
  - Prioritize alerts by importance.
- Idea: Enhance pattern matching by incorporating context
  - Start with (extended) traditional signatures.
  - Do not generate an alert for every match but analyze further.
- Context on different levels:
  - Low-level: Syntactic context, connection state.
  - High-level: Knowledge about network state.

Contextual Signatures (2)

- Open-Source NIDS Bro as platform
  - Deployed for example at UCB, LBNL and TUM.
  - Supports different detection approaches.
  - Abstracts network traffic into events.
  - Provides full scripting language to evaluate events.
  - Until now: signature matching possible but cumbersome.
- Contextual Signatures for Bro
  - Build new signature engine as a starting point.
  - On a match, raise an event rather than an alert.
  - Utilize already existing rich contextual state.

Example Applications

- Exploit Scanning
  - Aggregation of alerts reduces alert volume.
- Vulnerability Profiles
  - Knowing a system's specifics helps to prioritize alerts.
- Attacks with multiple steps
  - Recognizing sequence of steps increases accuracy.
- Request/Reply Signatures
  - Server's response may indicate result of attacks.

IIS Exploit Attempt

- Alert if server does not respond as expected.

```
signature cmdexe-success {
  ip-proto == tcp
  dst-port == 80
  http /.*cmd\.exe/
  requires-signature-opposite ! http-error
  tcp-state established
  event "WEB-IIS cmd.exe success"
}

signature http-error {
 ip-proto == tcp
 src-port == 80
 payload /.*HTTP\/1\.. *4[0-9][0-9]/
 tcp-state established
 event "HTTP error reply"
 }
```

Features of the Signature Engine

- Pattern matching uses full regular expressions
  - Expressive power.
  - High performance
    - Parallel matching in time linear in size of input.
    - But: Size of DFA may grow expontentially $\longrightarrow$ Build incrementally.
- Interfaces to Bro's already existing functionality
  - Connection state management (time-outs, bi-directionality).
  - Protocol analyzers (TCP, HTTP, FTP,... ...).
  - Scripting language (events, call-backs, identifiers).
- Leverage Snort's library by means of a converter.

Evaluation of the Signature Engine

- Evaluation by comparing against Snort
  - Convert ca. 1100 Snort signatures into Bro's language.
  - Run both systems on traces from different enviroments
    - 30-minutes full trace from U Saarbrücken (ca. 10GB).
    - 2-hours HTTP client-traffic from LBL (ca. 670MB).
  - Compare alerts and run-time.
- This does *not* evaluate the contextual signatures, but is a necessary first step to make sure the engine is usable.
- Overall results
  - Systems mostly agree on alerts and show comparable run-time.
  - More interesting: It is surprisingly difficult to compare two NIDSs.

Difficulties: Who’s correct?

- If they disagree, who is correct? *What* is correct?
- Different internal semantics
  - State management: Time-out based vs. memory-bounded.
  - TCP stream reassembly: Real stream vs. “virtual” packets.
  - Level of detail: Fine-grained vs. coarse-grained.
- How to configure the systems to compare the results?

Difficulties: Who's more efficient?

- Run-time depends on algorithms.
  - Bro analyzes much more carefully than Snort.
- Run-time depends on traffic
  - Small patch speeds up Snort by a factor of 2.6 for Web traffic.
  - Snort's serial matching can be twice as fast as the parallel (!).
- Run-time depends on hardware
  - Moving from P3 to P4, Bro gets faster while Snort gets slower.
- Bro vs. Snort: From 2 times slower to 3 times faster.

Summary

- Most deployed NIDSs use basic pattern matching.
- Contextual signatures consider more than bytes.
- Integration into Bro allows powerful applications.
- Leveraging Snort signatures provides a base set.
- It is hard to compare NIDs fairly.
- Future work: using contextual signatures operationally.

Enhancing Byte-Level Network Intrusion Detection Signatures with Context

Robin Sommer

`sommer@in.tum.de`

Technische Universität München
Germany

Vern Paxson

`vern@icir.org`

International Computer Science Institute
Lawrence Berkeley National Laboratory
Berkeley, CA, USA