

The Tao of Network Intrusion Detection

And How it Manifests in Firestorm

Gianni Tedesco

Overview of Firestorms Architecture

- Firestorm is a data-processing pipeline like any other.
- That is to say, network packets get stuffed in to one end, and alerts squirt out of the other end.
- Technical details:
 - It's about 30 KLOC (written in C)
 - Runs as a single-threaded process on any UNIX-like system.
 - Plugin architecture (DSOs)
 - Most core code is in a shared library
 - API/ABI is still unstable

Step 1. Data Acquisition (capture)

- Responsible for getting raw data from the outside world.
- 'capdev' plugins hide details such as:
 - Various packet capture APIs available.
 - Some are OS specific
 - We may wish to capture from networks, from files, from IPC, etc..
- BTW. Anything packetized will work (SS7/GSM)
- Some things that aren't packetized kind of work too...

Step 2. Decode

- Demultiplexes layered traffic.
- Each protocol is implemented as a plugin.
- Some protocols (TCP) break streams in to packets. We have to reconstruct the stream.
- That is non-trivial because we need to extract relevant sections of the streams for analysis.
- In fact, the problem at this stage is validating, uniformizing and disambiguating traffic in order to facilitate accurate analysis.
- I swear to god I didn't make those words up.

Step 3. Analyze

- Discard noise and only allow interesting traffic to proceed. That is to say, generate alerts.
- Signature matching is the predominant methodology. For some good reasons:
 - Humans use reasoning/rationalizing ability to create the signatures.
 - Determinism. When an alert is generated, it is easy to see why.
- In firestorm this part is pluggable too.
- So we could also be doing anomaly detection here using AIS/AI/Bayes/etc...
- Obviously, this is the bottleneck.

Step 4. Alert

- Log alerts created by previous section (analysis).
- Sensors have finite storage
 - That means we have to send data to the analyst and delete logs at some stage.
- Sensors may be deployed over different geographic locations.
 - That means we need to use telecommunications to send alerts to the analyst (Internet).
- Communications may be observed or disrupted by the enemy.

Step 5. Console

- Present alerts to the analyst in useful ways.
- There will be lots of alerts. So performance matters.
- There will be lots of false positives. So the analyst wants to filter alerts.
- One intrusion may be represented by multiple alerts. So the analyst wants to correlate and group alerts.

Recap

- Firestorm has five main subsystems, representing the five stages of the pipeline.
 1. Data acquisition
 2. Decode
 3. Analysis
 4. Alerting
 5. Console
- Firestorm is extensible at each stage.
- This is the tao of network intrusion detection.

It's Really Not That Simple(tm)

- Every stage has unsolved problems.
- To be expected - field is still young.
- Can't be solved entirely by software, there is a good element of hardware/wetware:
 - Sensor Placement
 - Sensor -> Console Comms. Infrastructure.
 - Procedures, management, training.
 - Skill of each individual analyst.
 - Boring stuff.... ;)

Getting Access to Data

- Physically getting access to required data can be a challenge.
- Span/Mirror ports on switches are not suitable.
- A 24 port 100 Mbps full-duplex switch has peak throughput of 1.2 Gbps and peak packet rate of 2.3 Mpps.
- Ideally use a tap next to a router. Routers are on the edge of security zones, and do layer 3 processing typically at 100 Mbps.

Data Loss

- If sensor cannot process traffic as fast as it comes, data will be lost.
- If NIC raises an interrupt for each packet (at 2.3 Mpps) then this is massive overhead. Solved by NAPI RX polling drivers on Linux.
- If OS has to context switch for each packet, then this is another huge overhead. Firestorm supports Linux mmap packet socket which mitigates this overhead.
- Linux mmap packet socket is also zero-copy.
 - Actually one-copy, but removing that last one isn't worth it (the overhead is the load to L2 cache after DMA xfer)

Data Loss (cont.)

- How does firestorm currently minimize overheads? Some examples:
 - TCP stream reassembly uses special $O(1)$ memory allocators, and self-optimizing data structures.
 - Signatures represented as n -ary decision tree.
 - Optimized setwise string search routines used for signatures.
 - Log output is buffered and can be shared between multiple spindles. Like RAID/0 but more robust.
 - As an example, a data-set that took 10 minutes to process 2 years ago, takes a little under 2 seconds today.
 - No assembly code yet! (MMX/SSE/AltiVec could help)

Accuracy

- Signature Accuracy. Not a huge problem if signature writer is given correct tools. Regex matching is not the correct tool. Full protocol decodes will make a big difference.
- Inaccurately decoding the data means an attacker can lie to the IDS and evade capture.
- TCP is the most important transport protocol today and we are still getting it wrong...

TCP Stream Reassembly

- If a stream is packetized, transmitted and then de-packetized at the other end, we had better make sure our stream looks the same as what the receiver saw. This is not simple for TCP given:
 - Ambiguities in the TCP standard itself.
 - Variations in implementations (standards conforming or not)
 - Our current inability to detect what TCP stack is running on the receiver.
 - Receiver may be discarding packets due to lack of memory or other reasons.

Brief Digression: Wiretaps

- Noone has mastered TCP reassembly yet.
- If anyone says they have, they are lying.
- Therefore we cannot accurately reconstruct TCP streams by sniffing packets.
- Of course, the enemy know this, and this is why he can bypass carnivore / internet wiretapping.
- Keep shtum until the govt. has already legislated, OK? ;)

Lack of Preemption

- How can we detect new threats not yet known to the signature writer / security community?
- Nobody knows..... :(
- Plenty of ideas but no truly compelling implementations yet.
- We may need to expand logging infrastructure to log extended information about why packets, or sequences of packets are flagged as alerts in order to avoid automation irony.

Data Loss in Logging Phase

- Alerting to existing formats like tcpdump/pcap files or worse, a line in the syslog, loses way too much data.
- We need to save alert information and full packet data.
- There are compelling practical reasons to want to save decode information too.
- Firestorm uses ELOG.
- When an alert occurs as part of a stream, we cannot log the whole stream without expending considerable resources.

Presenting the Information

- When retrieving data from sensors, communications can be intermittent.
 - Disk spools must be used.
 - We don't want to lose or duplicate data. That's difficult.
 - Remember, even the internet mail system can duplicate emails.
- The analyst needs to search and sort over loosely structured data. Usual approaches like throwing all the data in an RDBMS are way too inefficient.
- Firestorm uses the ELOG files directly, but indexes them for sort/search.

Future Directions For Firestorm

- Ongoing performance improvements
- Intelligent stream reassembly (for TCP)
 - Massively improves performance and memory usage
 - Keeps relevant parts of the stream together
- More application layer decoders
- Fully implementing 'stormwall'
 - System for transmitting ELOG spools to the console.
- Console
 - Lots of UI things.
 - Searching / Sorting / Grouping.
 - Sensor management.
 - Correlation / statistical analysis.

Thanks!

- Firestorm website is at
 - <http://www.scaramanga.co.uk/firestorm/>
- My website is at
 - <http://www.scaramanga.co.uk/>
- Further reading:
 - Ptacek, Thomas H., Newsham, Timoty N. Insertion, Evasion, and Denial of Service: Eluding Network Intrusion Detection. (January 1998)
 - Anything by Stephen Northcutt.
 - Sun Tzu. The Art of War.