cinderella: A Prototype For A Specification-Based NIDS

Andreas Krennmair
krennmair@acm.org

August 8, 2003

Abstract

What is actually network intrusion detection? How does it work? What are the most common methods? What's cinderella? What are cinderella's concepts? How does it behave with common attacks? How can it possibly be evaded?

Agenda

- Introduction to network intrusion detection
- Introduction to cinderella
- cinderella's concepts
- cinderella's behaviour
- Evading cinderella

“So, what’s actually this intrusion detection thingy?”

- Networks that are publicly accessible via the internet (e.g. SMTP and web servers) are under constant attack (mostly by script kiddies)
- Unusued ports and services that shouldn’t be publicly accessible are protected by packet filters
- Problem: publicly accessible services can’t be protected by packet filters
- Solution:
  - use secure software (DJBware *flame*, SE Linux, ...)
  - use sandboxing tools like systrace
- $\rightarrow$ intrusion detection is not about preventing attacks or protecting systems
- Network intrusion detection is about detecting attacks and benefiting from the information gained during the detection process

Warnings And Hints

- **Network intrusion detection systems aren’t turn-key solutions.** Any NIDS vendor asserting that is talking total crap.
- **Network intrusion detection systems can’t prevent attacks.** Any NIDS vendor asserting that is talking total crap.
- **Network intrusion detection is not about getting informed who attacked which host by using which exploit.** Current NIDS vendors and projects want to make you believe that, obviously that’s bullshit. Everyone who wants such a system should install a blinkenlight on his/her desk that blinks whenever an attack was detected.

Network Intrusion Detection Techniques

- Misuse detection
- Anomaly detection
- Specification-based detection

Misuse Detection

- NIDS has a set of patterns
- Tries to match these patterns against what is currently sniffed off the network.
- Disadvantages:
  - Attacks must be known before they can be detected
  - Unknown attacks can’t be detected
  - It’s very likely that variations of a known exploit can’t be detected (unless the patterns are very good, changing a few bytes in the exploit string can lead to evasion)
  - The set of patterns is going to grow $\rightarrow$ the larger the set, the longer it takes to match the patterns against the traffic

Anomaly-based Detection

- Uses machine-learning techniques, most often implemented using statistical methods
- The system is trained how “normal” (good) traffic has to look like
- All anomalies are marked as “abnormal” traffic
- Advantages:
  - Unknown attacks can be detected
  - No database of patterns has to be maintained
- Disadvantages:
  - System must be trained with basically the same or similar attack-free traffic that will be encountered in live operation
  - When too few features are trained, the system can’t be effective

Specification-based Detection

- “Good behaviour” is first defined programmatically
- Anomalies of this defined “good behaviour” are marked as bad
- Difference to anomaly-based detection: **no training**
- Better: the programmer “maps” his/her trained knowledge to the program
- Low rate of false alarms (at least several papers say so ;-)
- cinderella implements some of these concepts

Facts About cinderella (1)

- Prototype for a NIDS implementing specification-based detection (not evolutionary!)
- Basic concepts
  - “in dubio contra reo”: any traffic that is not clearly detected as good traffic is automatically (per definitionem) bad
  - Details about attacks are irrelevant: this has already been implemented by other projects, and won’t help us anyway
  - Keep the codebase small and simple
  - Keep cinderella modular and extensible
  - cinderella may act as “wrapper” around other NIDS

Facts About cinderella (2)

- Implemented using Ruby: Ruby is a pretty good prototyping language
- Should work on any platform supporting Ruby 1.6.x and ruby/pcap (Ruby module to interface with libpcap)
- Further goals: when cinderella is stable and mature, it shall be reimplemented in a compiled language (preferably C + libowfat) and algorithms and data structures shall be optimized
- Why the name cinderella?
  - "Cinderella" is the English version of the well-known Brothers Grimm tale "Aschenputtel"
  - This tale contains a famous quote that very well describes what cinderella (the NIDS) does: **"The good into the pot, the bad into the crop."**

“So, how does cinderella do this intrusion detection stuff?”

- Ethernet frames are sniffed off the network, packets other than IP are discarded
- TCP, UDP, ICMP and other packets are separated
- TCP packets are reassembled to the complete “stream” of the connection
- A connection is identified by `SrcIP:SrcPort` and `DstIP:DstPort`
- A “conversation” is built from the connection
- This conversation is evaluated by a module
- When the connection is identified as good or bad, it is marked as such, and written to a corresponding tcpdump file
- Else, the connection is reevaluated when the connection’s next packet arrives
- Connections that aren’t explicitly configured are automatically “bad”

Implementation Of UDP And ICMP

- UDP:
  - Implementation similar to the TCP modules
  - Easier than TCP, since only single packets have to be analyzed, and not whole connections
  - The modules for most UDP-based protocols can be implemented using stateless logic
- ICMP:
  - Policies instead of modules
  - Policies define which host(s) may send which ICMP codes to which host(s)
  - ICMP packets without corresponding policy are marked as bad

Example Configuration

- TCP modules:

```
tcp modules/tcp_http.rb ^.*$ ^192\.168\.124\.1[3-5]:80$
```

- UDP modules:

```
udp modules/udp_dns.rb ^192\.168\.\d+\.\d+:\d+$ \
                       ^192\.168\.123\.16:53$
```

- ICMP policies:

```
icmp ^192\.168\.\d+\.\d+$ ^.*$ echo,echoreply
icmp ^193\.170\.156\.1$ ^.*$ routeradvert,routersolicit
```

Example: HTTP Module (1)

```
class TcpModule

  def initialize
    goodlist_file = "etc/http_goodlist.txt"
    @regexps = Array.new
    IO.foreach(goodlist_file) do |line|
      line.chomp!
      @regexps << line
    end
  end
```

Example: HTTP Module (2)

```
  def evaluate(conv)
    if conv.size > 0 then
      @regexps.each do |re|
        if conv[0] =~ re then
          return [ true, true ]
        end
      end
      return [ true, false ]
    end
    return [ false, false ]
  end
end
```

Example: HTTP Module Configuration File

```
^GET /([a-z]/){0,3}[a-z0-9]{1,10}\.html HTTP/1\.[01]
^GET /cgi-bin/[a-z]+\.(pl|cgi)(\?[a-z&=]{0,20})? HTTP/1\.1
^GET /exploitable-cms/[a-z]+\.php3? HTTP/1\.1
```

Scripts to auto-generate these configuration files can be easily written (it has been already done within 50 lines of Perl code , for cinderella's non-free predecessor).

cinderella In Practice

- How does cinderella handle port scans?
- How do cinderella’s concepts make it possible to detect Code Red, Nimda and possible other similar worms in the future?

How Does cinderella Handle Port Scans?

- TCP connection attempts that are not explicitly defined in cinderella's configuration are marked as “bad”
- In case it is defined, but the connection is either reset or closed without sending any payload, the traffic is marked as “bad”, too
- Although port scans are not explicitly handled, they can be easily detected, thanks to “in dubio contra reo”
- In practice, packet filters are placed in front of a publicly accessible network to reduce unnecessary traffic and to reduce false positives that a NIDS could detect → port scans usually don't even reach the NIDS

How Do cinderella’s Concepts Make It Possible To Detect Code Red, Nimda And Possible Other Similar Worms In The Future?

- In the late summer of 2001, the two worms Code Red and Nimda struck the internet, exploiting a buffer overflow in some DLL of MS IIS
- Several variations of Code Red existed, with different padding characters (version 1: `N`, version 2: `X`)
- cinderella’s HTTP module only marks HTTP requests as “good” that are explicitly defined
- It is very unlikely that the NIDS’s administrator defines some worm’s exploit string as allowed HTTP request
- → Code Red, Nimda and similar worms are detected due to cinderella’s “in dubio contra reo” concept

cinderella As Wrapper

I mentioned it before: cinderella is intended to act as a wrapper around other NIDS. The reason for this is also the reason why I said before that network intrusion detection is not about immediately detecting attacks.

- cinderella is intended to separate suspicious from unsuspicious network traffic
- The unsuspicious network traffic is discarded
- Another NIDS (e.g. Snort) has a look at the traffic
- This other NIDS discards all traffic in which it finds exploit strings/known patterns
- What is left **should** contain previously unknown exploits and attacks

Drawbacks And Possible Evasion Of cinderella

Practical problems that currently make cinderella not really usable in production environments:

- IP fragment reassembly isn’t yet implemented. This is a crucial feature $\rightarrow$ specially-crafted fragmented IP packets could pass cinderella undetected
- The TCP state machine is working but not 100 % perfect
- Modules for a wide range of protocols are still missing (SSH, FTP (yuck), DNS, IMAP, NTP, ...)
- You are invited to try evading cinderella! I’m desparately waiting for your bug reports

Where Can I Get cinderella From?

- Source code:

  `http://developer.berlios.de/projects/cinderella/`

- Documentation/general information:

  `http://synflood.at/cinderella/`

Comments, Flames, ...

- Flames go to `/dev/null`, where they are recycled to a biodegradable bitstream that is reused as entropy by `/dev/random`
- All other comments go to `ak-cinderella@synflood.at`

Any Questions?

You can ask me now or after the lecture. Again, the most important addresses:

```
http://developer.berlios.de/projects/cinderella/
http://synflood.at/cinderella/
mailto:ak-cinderella@synflood.at
```