

# **Realtime Intrusion-Forensics**

## **A First Prototype Implementation**

*(based on a stack-based NIDS)*

Udo Payer

Institute of Applied Information Processing and Communications  
University of Technology, Inffeldgasse 16a, A-8010 Graz, Austria

email: [Udo.Payer@iaik.at](mailto:Udo.Payer@iaik.at)

### **Abstract**

*The function of a Network Intrusion Detection System (NIDS) is to identify any misuse and abnormal behavior determined as an attack to a network segment or network host. The proposed concept is a pump-in-the-stack approach. This means, that NIDS-features are integrated into the network stack of our operating systems.*

*Using the native stack is important, since this is the only place in our operating systems where we can get access to all packets (passing the stack) in realtime quality. The idea is to make use of already existing knowledge about state transitions, memory content, header information, and packet payload. This is very similar to stack hardening. But while hardening mechanisms are limited to block malicious traffic (violating RFC793), the proposed approach is to collect as much evidence as possible and to do some simple forensic analysis. Knowing that IPv4 is not suitable to collect information about the actual source of an attack, but there is no real difference to traditional IPv4 based forensic analyses.*

*In addition to simple stack hardening mechanisms, the advantage of the proposed approach is to start forensic analysis long time before the host is going to become a "pathologic case".*

*Maybe that collecting forensic evidence and the preservation of collected information is inappropriate in the case of intrusion detection systems (IDS). But IDSs are the most likely candidates (at least in the absence of alternatives) to collect forensically pristine evidentiary data, if real- or nearly realtime behaviour is required [1].*

*To verify this statement, two prototypes were built (representing the two most popular categories of operating systems) and stack-based intrusion detection mechanisms have been integrated into the network stack.*

## **1 Introduction**

A Network Intrusion Detection System (NIDS) is a system that is designed to detect abnormal or not expected data that is the result of an unauthorized event in the network. This is unlike a firewall, which is configured to allow or deny access to particular services or hosts (*based on a set of rules*). While misuse-based NIDSs are more or less limited to the existence of a corresponding signature, anomaly detection is restricted to heuristic methods (*due to the lack of clear defined rules and guidelines*). Different from these classical approaches, a stack-based stateful mechanism can be used to introduce "intelligent" decisions in searching conspicuous patterns.

Unlike the first two classical approaches, stateful inspection can track each connection traversing all network stacks.

Today, a great number of different stateful approaches exist. But in the case of stack-based mechanisms, the proposed approach can even be used in realtime environments.

Realtime behaviour always requires strict timing constraints and therefore small signatures and fast scanning mechanisms. Thus, the proposed scanning engine can be considered as a simple state based filter using small signatures (*based on simple state transitions or sequences of state transitions*).

## 2 State-Transition Signature Recognition

Since the problem of signature recognition is caused by the great number of database entries, the state-based approach tries to limit the number of entries by defining signatures as a dedicated sequence of state transitions. Since state machines already exist in our network stacks, the states (*rather the transitions between two states*) can be used as indicators for intrusion evidences. In addition, a single signature can be applied to similar or allied intrusions. Therefore unique state transition sequences can be defined and stored as single database entries. Furthermore, the verification of a state-based signature (*a single intrusion or a group of similar intrusions*) can be executed efficiently and even in parallel. Searching for string based signatures (*in contrast*), can only be done in serial.

The idea of state driven intrusion detection is not new. K. Ilgun describes in [5] a rule-based penetration identification method based on maintaining a set of state transition signatures. If a specific sequence is found, an intrusion is alarmed. The main goal is to analyse the state transition behaviour rather than auditing the containing information. Today, a great number of similar papers describe mechanisms, based on the same state transition approach [7].

But whereas literature on state transition based IDS is limited to the specification of a formal description language [2], tools to model network attacks [4], and analysis techniques [8], the proposed approach has its main focus on the exploitation of already existing state machines and their state transitions at any network layers. Therefore, the main goal of the proposed approach is the ability to integrate state based detection mechanisms into our network stack:

*The network stack state machines of our operating systems should be reused to perform detection mechanisms. Intrusion detection should be an inherent service causing no extra charges.*

Protocols up to the application layer can also be treated in the same way by taking advantage of the application protocol state machines and state transitions due to application events.

Based on this state-driven and layered NIDS-approach, the next sections will describe how to make use of this mechanism, and how to find signatures, capable to collect forensic evidences.

## 3 Thoughts on Online Forensic Investigations

The concept of combining evidence gathering with system protection is not new and has initiated several discussions in the past. The problem is that it is unsure that network security services are appropriate tools to collect evidences during ongoing attacks. But whereas classical forensic analysis is limited to “post-mortem” analysis (*of already pathological victims*), a realtime forensic analysis can give us a theoretical chance to find some evidences about —*maybe more intrusive*— next steps. Intrusive attacks can be classified on-the-fly and appropriate countermeasures can be started, since simple attacker-profiles can be derived from recorded malicious traffic.

Classic forensic analysis always starts with a search through the sources of evidence (disc drives, log files, boxes of removable media,...) and tries to perform two things:

- To make sure to preserve as much data in its original form, and to
- re-construct the events that occurred during a criminal act and to produce a meaningful starting point for police and prosecutors to do their jobs.

Similar mechanisms can be applied to “*online forensic analysis*” but these mechanisms always require the presence of host-based intrusion detection systems, since this is the only way to get direct access to any host-based information. But the target of the proposed approach is not to achieve the same quality like “*classical forensic analysis*”. The main idea is to use the short-time history of currently used connections to generate simple attacker profiles.

People do not only feel confident about this idea. Sommer [10], reports that the NSTAC Network Group - Intrusion Detection Subgroup found in December 1997 that:

- Current intrusion detection systems are not designed to collect and protect the integrity of the type of information required to conduct law enforcement investigations, and that
- there is a lack of guidance to employees as to how to respond to intrusions and capture the information required conducting a law enforcement investigation.

Contrariwise, Yuill et al [11] claimed that intrusion detection systems are able to collect enough information during an ongoing attack to profile the attacker.

It is important to remember that the steps in preserving and collecting evidences should be done very carefully, methodically, and deliberately. The various pieces of data (*the evidence*) on the system are telling the story of what occurred. The *first instance*, capable to respond has the responsibility to ensure that as little of this evidence is damaged, thereby making it useless in contributing to a meaningful reconstruction of what occurred [11].

*This very first instance is the operating system. Humans usually cannot react in appropriate time and (usually) do not perform the required steps of preserving forensic evidences methodically.*

The job of a forensic investigator, is to do the best to search through the sources of evidence and to perform two things: (1) make sure that as much of this data remains in its original form, and (2) to try to re-construct the events that occurred during a criminal act and to produce a meaningful starting point for police and prosecutors to do their jobs.

## 4 Strategy/Tactic

As soon as we have recognized, that our system is under attack, we have to understand all methods, an intruder can use to gather some information about our system.

The primary job of an online forensic investigator is to preserve as much evidentiary data at the crime scene and in any usable form. In doing so, evidentiary information can be a sequence of network packets (*used for OS detection*), malformed packets, and packets from forged sources (*inasmuch as it can be proved in IPv4*).

To verify the statement of Yuill et al [11], a set of suitable detection mechanisms was implemented (*based on two representative operating system network stacks*). This set of suitable detection mechanisms is described in the following:

### 4.1 IP Spoofing

IP spoofing is the most popular mechanism to hide the trace of an attacker, or is part of an indirect attack (e.g. *SMURF attack*). But there are some very simple methods to recognize the fact that somebody is trying to spoof an IP address. The best way to detect this kind of attack is to monitor the reply to the SYN+ACK packet (sent by the victim). If the spoofed host is still alive it will return a NACK (RST flag set) which can be used to identify this type of attack. Another method is to check the acknowledgement number arriving right after the SYN+ACK was sent. If the acknowledgment number doesn't comply with the initial sequence number, the attacker is trying to guess the ISN.

*Due to the problems with IPv4, IP-spoofing detection is the basic element of uncertainty in classifying the quality of collected forensic evidences.*

All this can be executed together with active countermeasures. Such a countermeasure can be a simple packet which is sent to the spoofed host - just to make sure that the host is alive and is not supposed to DoS or DDOS attacks. The significance of this information can be improved by packet tracing. And as soon as the host is answering correctly, we can be sure that everything is alright. Otherwise, this information can be stored and used as forensic-evidence.

### 4.2 OS detection

Remote operating system fingerprinting is the process of determining the identity of the remote host's operating system and to guess the version number of the underlying kernel or services. This is done by sending packets to the remote host and by analysing the responses. Tools like Nmap and Xprobe2 are based on these responses and are using fingerprints which can be queried against a signature database of known operating systems responses.

Nmap (for instance) works best when Nmap can find at least one open TCP port, one closed TCP port, and one closed UDP port. Nmap works by sending a well-known set of test packets (*TSeq*, *T1-T7*, *PU*) against the target machine to determine the operating system. Fortunately, the sequence of test packets as well as the common behaviour of Nmap are known and can be detected by using sequences of state transitions. These known sequences can be treated as a single state transition based signature:

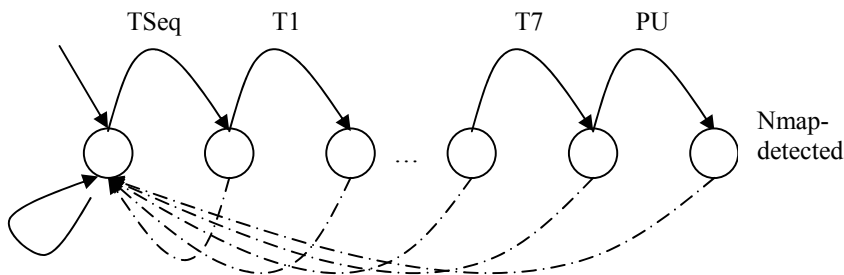


Figure 1 State transition based Nmap-signature

Nmap offers many “advantages” compared with other active OS fingerprinting tools, since half-open connections can be used to enable stealth scans, which will not be logged by standard IDS. But fortunately, it is very simple to respond to this kind of attacks, if direct access to the network stack exists.

A very effective countermeasure against Nmap is the changing of the TCP/IP stack behaviour. This can be accomplished by sending a single RESET, if a non-SYN packet was received on an open port by a UNIX workstation. On Windows platforms, we just have to prevent the sending of RESET in the same case (realized by deleting a single line in the *TCPIP.sys* source code).

```
// If it doesn't have a SYN (and only a SYN), we'll send a reset.  
// the following lines have to be deleted ...  
  
Unfortunately, the real source code is not allowed to be printed here ;-)
```

These small changes simulate Unix-like behaviour on Windows machines (and vice versa) and can successfully confuse OS fingerprinting attacks.

Since Nmap can be detected very easily, the different ways an attacker is using this tool can give us a lot of information about the skills and intentions of the intruder. Thus, this information is very important in terms of forensic analysis and can be used to rate the vulnerability of subsequent attacks.

But there are also other tools existing, based on completely different approaches. Xprobe2 for instance does not run port-scans against the target machine. Xprobe2 is heavily using the results found in the “*ICMP Usage in scanning*” research project. But due to the fact that the basic mechanism is known, the behaviour of such OS detection attacks can be detected as well.

### 4.3 Blinding the Network Stack (DoS or Stick attack)

Attacks, qualified to defeat signature based NIDSs can also be used to mask real attacks. In principle, always the same mechanisms are used to bring NIDSs to their knees. All these mechanisms are mainly based on extremely small-, fragmented-, malformed- or combinations of these “unusual” packets.

The problem with most NIDS sensors is that they do not completely reassemble fragmented packets. But by using the native network stacks, it is obvious that performing a complete defragmentation is not a limiting factor.

Some other attacks —like the popular “stick-attack”— were able to blind most of the industry leading NIDSs. Even if the stick attack is simply based on the well known SYN-Flooding attack, many NIDSs were not prepared to be immune against this simple mechanism.

But in the proposed approach, we can react on many of these known attacks to improve the effectiveness of “advanced” NIDSs on top of the network stack.

### 4.4 Shell Code- and Polymorph ShellCode Detection

Since “*Smashing the stack for Fun and Profit*” (a great paper on buffer overflows, written by Aleph One) is widely known, anyone is able to make use of one of the most insidious data-dependent bug to mankind [12]. To confine the threat of this serious attack, simple mechanisms (based on pattern recognition) are adequate means to search for evidences like NOP-zones (sequences of 0x90 on Intel machines) or known shellcode patterns (“/bin/sh”).

Mechanisms like this can be integrated anywhere in our network stacks wherever we have access to the packet-payload. But in terms of network forensics, we always need to know the relationship between suspicious contents and their alleged sources. Since IP-addresses are once more the only alternative to find a relationship between forensic evidences and the real source of suspicious packets, the network stack is once again the only suitable place, to start active countermeasures to analyse IP spoofing attempts.

Unfortunately, it is not very easy to recognize malicious packet content, since a technique called “*polymorphism*” appeared in 1992. The idea is very simple and can turn the shellcode problem in a really serious one. By simply ciphering the shellcode and attaching a decipher routine, the code can permanently change. Thus, it is not possible to use the same signature based mechanisms as described before.

A simple instruction call

```
execve("/bin/bash", ["/bin/bash", null], null)
```

is normally coded as

```
\x6A\x68\x68\x2F\x62\x61\x73\x68\x2F\x62\x69\x6E\x89\xE3\x31\xD2\x52\x53\x89\xE1\x6A\x0B\x58\xCD\x80"
```

but can be modified and embedded in a polymorph frame, looking like the example below[17].

FAKENOP	decipher routine	encrypted shellcode	bytes to cram	return address
---------	------------------	---------------------	---------------	----------------

This mechanism and all its “*advantages*” compared to ordinary shellcode attacks are described by Theo Destrian et al [13]. But he also gives some hints about the weak points of this mechanism. One of the most interesting one is the fact that this method cannot exist without a NOP-zone - even if methods are used to hide simple NOPs by one- or two-byte instructions. If one-byte instructions are used, the FAKENOP part can be detected very easily. Another way is to hide one-byte instructions in an alphanumeric zone [13]. But even this is not very efficient, if it is known that the corresponding service does not expect alphanumeric input. Another simple idea is the use of two-byte instructions, whereas each second byte- is a one-byte instruction or the first byte of a two-byte instruction (*and recursively*) [13]. Beside some other problems with this approach, a NIDS can make use of this systematic FAKENOP byte stream. Whenever different sequences are expected, this behaviour can be detected by using the state based approach:

By looking at the string `\x15\x11\xf8\xfa\x81\xf9\x27\x2f\x90\x9e` we can read [17]:

```
ADC $0x11F8FA81 #instruction demanding 4-bytes argument
STC #one-byte instructions
DAA
DAS
NOP
SAHF
```

... or by starting from the second byte

```
ADC %eax,%edx
CMP %ecx,$0x272F909E
```

This is a good example to show that byte-sequences can be found - always resulting in a valid sequence of instructions - independent from the starting point of the executable byte stream. These byte-sequences are self-aligning with the starting address of the subsequent deciphering routines. It's obvious, that this conspicuous behaviour can be detected by using simple state-machines:

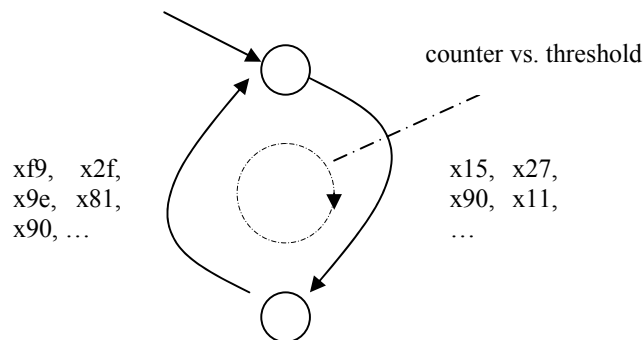


Figure 2 State transition based two-byte signature

This state machine is far from being complete and just an example to demonstrate the idea of implementing a state based one- and two-byte detection mechanism. Due to this simple approach, the integration into the network stack is practicable and will not decrease the over-all performance of our stack.

In terms of collecting forensic evidences, a mechanism to prevent shellcode execution is of immense interest, since this is the last bastion to defend the most serious threat.

If it is not possible to defend this “last attack”, traditional forensic analysis is going to replace online forensic analysis.

## 5 Results and To-Dos

Since all attacks can be detected by scanning TCP-flags, observing received packets and their correctness in terms of RFC793, or by applying filter rules to the content of a single or fragmented packet, the additional code is limited to tens of code lines. Thus, no measurable delay in packet processing can be detected. This is a nice side effect, but the timing behaviour of the whole network stack has to be concealed anyway to defend time based OS-fingerprinting attacks.

Since weeks or month of monitoring can produce large amount of data, the resulting log files cannot be locally stored for such long periods. Therefore, some sort of preselection has to be performed automatically.

Furthermore, simple countermeasures are started automatically – at least to confirm the existence of an ongoing attack (*IP spoofing detection*). Another nice side effect is the small number of additional code for these countermeasures. Since, placing the code at the right location in the network stack can reduce the effort for active countermeasures to a minimum.

The main problem of the proposed approach is the long-time storing of forensic evidences. It is obvious that advanced analysis mechanisms (*based on collected intrusion evidences*) are not allowed to be executed on the same machine. Therefore, we have to solve the problem of transmitting forensic information to a centralized server. Whereas the simple transmission is solved by [14][15][16] and [17], we still have the problem, that someone can use this mechanism to start a DoS attack by sending a small number of malicious packets - causing a huge number of intrusion detection messages. Another problem is the loss of information (*just stored on the sensor itself*) if someone succeeds in simple running DoS attacks against the centralized server. Unfortunately, all these problems are not solved so far and needs some further investigation.

Another crucial point is the problem to convince the OS-producer to integrate intrusion detection features into their network stacks.

While SYN-cookies (*Linux*) and stack-tuning (*Windows2000, XP*) are first steps in the right direction, I do not think that extended security mechanisms will be integrated into the network stacks in the foreseeable future.

## 6 Conclusions

Intrusion detection systems add an early warning capability to our defences, by sending alerts on any type of suspicious activities that typically occurs before or during an attack. Since most IDS cannot interrupt an ongoing attack, IDS should not be considered as an alternative to traditional security practices (*so called “rock-solid facilities” like firewalls and cryptography*). And they are no alternatives for a carefully designed security policy, backed up by effective security procedures which are carried out by skilled staff using the necessary tools. Instead of that, IDSs should be viewed as an additional mechanism, surrounded by a rock-solid perimeter protection.

Even though a lot of research work was done in the scope of intrusion detection, we are far-off the perfect solution. Therefore, new ideas should permanently be published and discussed.

This paper is a contribution to discuss the possibility of integrating forensic analysis into stack-based NIDSs. Another idea is the integration of state-based detection mechanisms into already

existing state-machines (*available in all our operating system network stacks*). This approach may lead to smaller footprints, and gives us the possibility to interfere in realtime.

The logic consequence is the introduction of intrusion prevention mechanisms and finally the collecting of forensic evidences. To use these mechanisms in realtime environments, they have to be integrated into the network stack of our operating systems.

Two fast prototypes were built (representing the two most popular OSs) to verify the basic concept of the proposed approach. At present, the process of finding a suitable set of adequate signatures is not completed, since it is a complex task to find an optimal set to cover most of standard attacks. But some nice demonstrations can be given to show the principle correctness of this approach. But the set of intrusion signatures (especially in the scope of forensic analysis) is still far from being optimal.

## References

- [1] P. Stephenson "The Application of Intrusion Detection Systems in a Forensic Environment", Proceedings of the RAID 2000 Conference, Toulouse, France, 2000.
- [2] S.T. Eckmann, G. Vigna, and R.A. Kemmerer, "STATL: An Attack Language for State-based Intrusion Detection", Journal of Computer Security, vol. 10, no. 1/2, pp. 71-104, 2002.
- [3] S. Kumar, "Classification and Detection of Computer Intrusions", Dissertation, Purdue University, August, 1995
- [4] G. Vigna R. A., "NetSTAT: A Network based Intrusion Detection System", Journal of Computer Security, 7(1), IOS Press, 1995
- [5] K. Ilgun, "USTAT: A Realtime Intrusion Detection System for UNIX", Proceedings of the 1993 IEEE Symposium on Research in Security and Privacy, May, 1993 pp16-28.
- [6] K. Ilgun, R.A. Kemmerer, and P.A. Porras, "State Transition Analysis: A Rule-Based Intrusion Detection Approach", IEEE Transaction on Software Engineering, 21(3), March 1995.
- [7] P. A. Porras and R.A. Kemmerer, "Penetration State Transition Analysis - A Rule-Based Intrusion Detection Approach", Proceedings of the eighth annual Computer Security Applications Conference, San Antonio, TX, 1992. pp220-229.
- [8] P. A. Porras, "STAT - A State Transition Analysis Tool for Intrusion Detection" - M.S. Theses, Computer Science Dep., University of California Santa Barbara, June 1992.
- [9] R. Spangler, "Analysis of Remote Active Operating System Fingerprinting Tools", May 2003
- [10] P. Sommer "Intrusion Detection Systems as Evidence", Recent Advances in Intrusion Detection - RAID 98
- [11] J. Yuill, S. Felix Wu, Fengmin Gong, Ming-Yuh Huang, "Intrusion Detection for an On-Going Attack", 2<sup>nd</sup> International Workshop on Recent Advances in Intrusion Detection - RAID 99
- [12] Aleph One, "Smashing the Stack for Fun and Profit", Phrack 49, Volume Seven, Issue Forty-Nine.
- [13] Theo Detristan, Tyll Ulenspiegel, Yann Malcom, Mynheer Sperbus von Underduk "Polymorphic Shellcode Engine Using Spectrum Analysis", Phrack 49, Volume Eleven, Issue Sixty-One
- [14] "Intrusion Allert Protocol" draft-ietf-idwg-iap-01.txt September, 2000
- [15] Intrusion Detection Message Exchange Requirements draft-ietf-idwg-requirements-10 October, 2002
- [16] The Intrusion Detection Exchange Protocol (IDXP) draft-ietf-idwg-beep-idxp-07, 2002
- [17] Intrusion Detection Message Exchange Format Data Model and Extensible Markup Language (XML) Document Type Definition, draft-ietf-idwg-idmef-xml-10.txt, 2003