Optimization of Network Intrusion Detection Processes

Von der Fakultät für MINT – Mathematik, Informatik, Physik,
Elektro- und Informationstechnik
der Brandenburgischen Technischen Universität Cottbus–Senftenberg
zur Erlangung des akademischen Grades
Doktor der Ingenieurwissenschaften
(Dr.-Ing.)

genehmigte Dissertation
vorgelegt von

Master of Science (M.Sc.)

René Rietz
geboren am 15.07.1982 in Cottbus

Gutachter: Prof. Dr.-Ing. habil. Hartmut König

Gutachter: Prof. Dr.-Ing. habil. Falko Dressler

Gutachter: Prof. Dr.-Ing. Felix Freiling

Tag der mündlichen Prüfung: 17. November 2017

Abstract

Intrusion detection is a concept from the field of IT security. Network intrusion detection systems (NIDS) are used in addition to preventative measures, such as firewalls, to enable an automated detection of attacks. Network security threats often consist of multiple attack phases directed against various components of the network. During each attack phase, varying types of security-related events can be observed at various points in the network. Security monitoring, however, is nowadays essentially limited to the uplink to the internet. Sometimes it is also used to a limited extent at key points within a network, but the analysis methods do not have the same depth as at the uplink. Other areas, such as virtual networks in virtual machines (VMs), are not covered at all, yet.

The aim of this thesis has been to improve the detection capability of attacks in local area networks. With a glance at the area of safety engineering, it appears efficient to secure these networks thoroughly and to develop additional monitoring solutions only for the remaining problem cases. This entails several challenges for the analysis of different parts of the TCP/IP stack. The lowermost part of the network stack has to be analyzed for attacks on network components, such as switches and VM bridges. For this, there is still no technology. Although attacks on the layers 2 and 3, such as ARP spoofing and rogue DHCP servers in physical networks, can be controlled to some extent by appropriate switches, equivalent methods are not used in virtual networks. Therefore, a software-defined networking based approach is proposed to counteract the respective attacks, which works for physical and virtual networks. The upper layers are already largely covered by traditional NIDS methods, but the rapidly increasing data rates of local area networks often lead to an uncontrolled discarding of traffic due to overload situations in the monitoring stations. Therefore, the drawbacks of current optimization approaches are outlined based on a detailed performance profiling of typical intrusion detection systems. A new approach for parallelizing the intrusion detection analysis that copes with the increasing network dynamics is introduced and evaluated. Since further special issues for NIDS particularly go back to the massive use of web technologies in today's networks, a firewall architecture is presented which applies novel NIDS methods based on machine learning to identify web applications and to ward off malicious inputs. The architecture addresses the entire process chain starting from the data transfer with HTTP via the analysis of manipulated web documents to the extraction and analysis of active contents.

Contents

List of Figures

List of Tables

List of Abbreviations

AJAX	Asynchronous JavaScript and XML
ALG	Application-level gateway
API	Application programming interface
APT	Advanced persistent thread
ARP	Address resolution protocol
ASCII	American standard code for information interchange
ASIC	Application-specific integrated circuit
ASLR	Address space layout randomization
AST	Abstract syntax tree
CFI	Control flow integrity
CFS	Completely fair scheduler
CPI	Code pointer integrity
CPU	Central processing unit
CSP	Content security policy
CSRF	Cross-site request forgery
CSS	Cascading style sheets
DARPA	Defense advanced research projects agency
DHCP	Dynamic host configuration protocol
DIDS	Distributed intrusion detection system
DDOS	Distributed denial-of-service
DMZ	Demilitarized zone
DOM	Document object model
DOS	Denial-of-service
DNS	Domain name system

DS	Directory services
DSSSL	Document style semantics and specification language
DPI	Deep packet inspection
DTD	Document type description
E	Error rate
EAP	Extensible authentication protocol
EAPoL	Extensible authentication protocol over local area networks
EBP	Extended base pointer
EPMAP	End-point mapper
ESP	Extended stack pointer
F1	F-measure
FMEA	Failure modes and effects analysis
FMECA	Failure modes, effects, and criticality analysis
FN	False-negative
FNR	False-negative rate
FP	False-positive
FPR	False-positive rate
FTP	File transfer protocol
GPU	Graphic processing unit
HTML	Hypertext markup language
HTTP	Hypertext transfer protocol
IANA	Internet assigned numbers authority
ICAP	Internet content adaptation protocol
ICMP	Internet control message protocol
ID	Identification
IDES	Intrusion detection expert system
IDS	Intrusion detection system
IIS	Internet information services
IMAP	Internet message access protocol

IMP	Inspection and modification protocol
IO	Input/output
IP	Internet protocol
IPS	Intrusion prevention system
ISP	Internet service provider
IT	Information technology
JS	JavaScript
L1/L2/L3	Layer 1/2/3
LAN	Local area network
LB	Load balancer
LDAP	Lightweight directory access protocol
MAC	Media access control
MIME	Multipurpose internet mail extensions
MS	Microsoft
MTU	Maximum transfer unit
NAT	Network address translation
NFS	Network file system
NIDS	Network intrusion detection system
ND	Neighbor discovery
NIC	Network interface card
NOS	Network operating system
NS	Neighbor solicitation
NSA	National security agency
NSM	Network security monitor
OFP	OpenFlow protocol
OS	Operating system
P	Precision
P2P	Peer-to-peer
PC	Personal computer

PCAP	Packet capture
PEAP	Protected extensible authentication protocol
PIC	Position independent code
PIE	Position independent executable
PNP	Plug and play
POP3	Post office protocol 3
RA	Router advertisement
RADIUS	Remote authentication dial-in user service
RAM	Random-access memory
ROP	Return-oriented programming
RPC	Remote procedure call
SDN	Software-defined networking
SGML	Standard generalized markup language
SLLA	Source link layer address
SMB	Server message block protocol
SNMP	Simple network management protocol
SOAP	Simple object access protocol
SPMD	Single-program/multiple-data
SQL	Structured query language
SSH	Secure shell
STP	Spanning tree protocol
SVM	Support vector machine
SYN	Transmission control protocol synchronize message
TCAM	Ternary content addressable memory
TCP	Transmission control protocol
TFTP	Trivial file transfer protocol
TLB	Translation lookaside buffer
TLS	Transport layer security
TN	True-negative

TNR	True-negative rate
TS	Terminal services
TP	True-positive
TPR	True-positive rate
UDP	User datagram protocol
ULA	Unique local address
URI	Uniform resource identifier
URL	Uniform resource locator
USB	Universal serial bus
VNC	Virtual network computing
VLAN	Virtual local area network
VM	Virtual machine
VOIP	Voice over internet protocol
W3C	World wide web consortium
WEBDAV	Web distributed authoring and versioning
WEP	Wired equivalent privacy
WPA	Wi-Fi protected access
WPS	Wi-Fi protected setup
WLAN	Wireless local area network
XHTML	Extensible hypertext markup language
XML	Extensible markup language
XSL	Extensible stylesheet language
XSLT	Extensible stylesheet language translation
XSS	Cross-site-scripting

1 Mitigation and Detection of Network Security Threats

Network security threats often consist of multiple attack phases against various components in the network. During each attack phase, varying types of security-related events and different traffic patterns can be observed at various points in the network, e.g., at the uplink to the internet, on switches in a local area network, or between hosts on a local network segment. The security monitoring, however, is nowadays essentially limited to the uplink to the internet. Sometimes it is also used to a limited extent at key points within a network, but the analysis methods do not have the same depth as at the uplink and some areas, such as virtual networks in virtual machines are even not yet covered. A holistic network security monitoring would have to cover the entire area from the internet uplink via all internal networks up to the virtualization hosts. In principle, a specialized intrusion detection system could be written for this purpose for each single subsystem. In the long term, however, it is not useful constantly chasing after the attack techniques for each network technology. With a glance at the area of safety engineering, it appears more efficient to secure networks thoroughly and to develop additional monitoring solutions only for the remaining problem cases. For this purpose, a security engineering process is required which is discussed in the next sections of this chapter.

1.1 Safety Engineering, Security Engineering and the Concept of Intrusion Detection

Intrusion detection is a concept from the field of IT security. Security is very closely linked to the field of safety. In some languages – for example German – no difference is made between the concepts of safety and security in the vocabulary. It is therefore obvious to adopt approaches from the better defined safety engineering for solving IT security problems. Safety engineering is a relatively well understood discipline which ensures by means of *failure modes, effects, and criticality analysis* (FMECA [4, 1]), as well as fault-tree analysis [5] that a system poses no risk to itself, the environment, or other systems. FMECA is a *process* that documents and evaluates all failure modes of a system. This includes an analysis of the failure impact on higher-level systems and the *identification/detection* of these errors during design, operation, or maintenance of the system, as well as *response/mitigation* strategies.

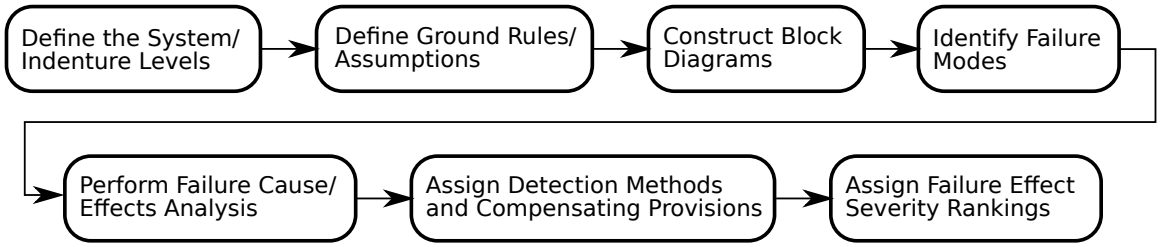

Figure 1.1: FMEA workflow (based on [1])

Figure 1.1 represents the FMEA process without criticality analysis. The first step of the FMEA process (*define the system/indenture levels*) defines the system with its individual functions and their interdependencies at several levels – typically the hardware broken into several parts and the software running on it. Potentially occurring errors are determined for each individual (hardware/software) part as well as their impact on interfaces to other parts. The next step is the definition of *ground rules and basic assumptions*. This is essentially a compilation of the function of the analyzed system/part, the expected operating time and life expectancy, the working phase in which a problem may occur, the severity of an error, the derivation of probability distributions for the error (manufacturer's data, statistical analysis, expert opinion), and error detection methods. The third step – the *construction of block diagrams* – is actually part of the previous step to (graphically) explain the individual components of the system and to ensure a tracing of observed errors through all levels of a system to their origin. The fourth step – the *identification of failure modes* – describes the manner how an error can occur within a component. There are many ways why a (physical) system can fail, e.g., overheating in a turbine or a pressure drop in the supply line to the lubricant may indicate the same error – loss of lubricant. The first part of the following fifth step – *failure cause and effects analysis* (failure causes) – is therefore to reduce the many failure modes to individual causes. The second part (failure effects) deals with the reproduction of the failures through all levels of the system. The sixth FMEA step – *the assignment of detection methods and compensating provisions* – deals with methods to detect the identified errors, e.g., sensors, and the error signaling to the operator of the system, e.g., audible or visual warning systems based on limits of measured sensor values. Compensatory provisions are typically redundant system parts that can take over functions in case of an error. The last step – *the assignment of failure effect severity rankings* – assigns the identified errors in a ranking according to their impact on the overall functionality of the system.

Currently there is no similar concept as FMEA in the area of IT security. However, some of the individual phases of such a process are applied in a similar manner. IT security primarily engages in attacks on the availability, integrity, and confidentiality of data. Therefore, the best correspondence to failures in the safety area is the loss of availability, integrity, and confidentiality of data. Figure 1.2 represents a possible

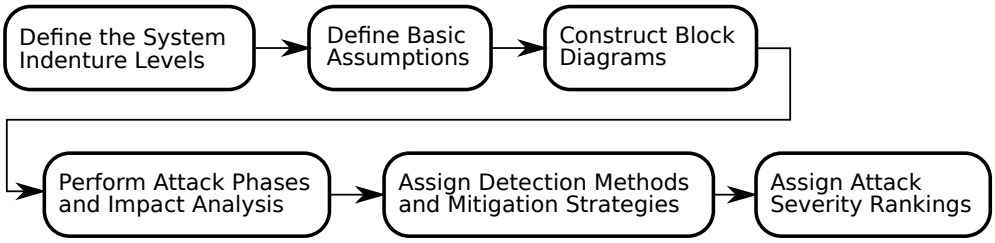

Figure 1.2: Potential Security Engineering Workflow

security engineering workflow based on the ideas from the safety engineering. The first three steps – the definition of the system indenture levels, collection of basic assumptions, and construction of block diagrams – can also be applied in IT security engineering, with the constraint that information processing systems are considered. This means that during the dissection of a system in its parts in particular interfaces for information processing are considered. The loss of availability, integrity, or confidentiality does not occur by itself but is the result of an attack. An analysis of potential attacks and their impact must be carried out in the next two steps instead of analyzing the failure cause, failure mode, and failure effects. The last two steps – the assignment of detection methods and mitigation strategies and the attack (effect) severity ranking – correspond to the actual core of an intrusion detection process. According to Edward G. Amoroso, intrusion detection is defined as follows:

> Intrusion detection is the *process* of *identifying* and *responding* to *malicious activity* targeted at *computing and networking resources*. [6]

Since the intrusion detection process itself is part of a larger security engineering process, this chapter first describes the surrounding process to enable a better understanding of how intrusion detection systems work. The following sections provide an overview based on the last three parts of the hypothetical engineering process of Fig. 1.2 starting with attack vectors and their impact via existing detection and mitigation strategies toward the attack severity rankings.

1.2 Attack Phases and Their Impact

An attack on an IT system usually consists of more than one attack phase. Attacks and their phases can be distinguished in long-term *targeted attacks* launched by organizations in combination with *Advanced Persistent Threats (APT)*, *attacks by individuals*, and *automated attacks*. To select countermeasures that can mitigate the effects of the individual phases, each of them must be examined more closely. In this section

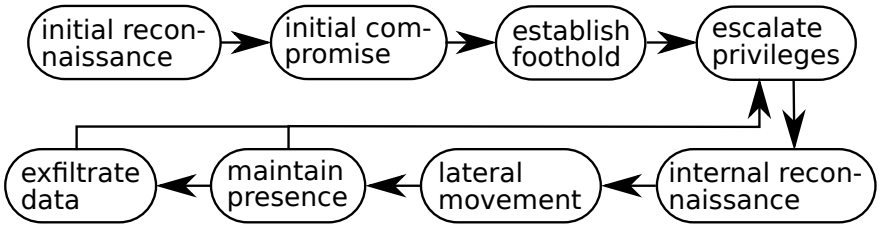

Figure 1.3: APT lifecycle (based on [2])

mainly targeted attack phases are discussed because they represent a superset of the automated and individual attacks.

For long-term targeted attacks, attackers proceed typically as follows [2] (cf. Figure 1.3). First in an exploration phase (*initial reconnaissance*), an attempt is made to find entry points into the destination network. This phase usually involves observable network scans which can be detected by network intrusion detection systems (see next section). The subsequent phase of *initial compromise* attempts to provide access to the target network. This can be achieved through direct attacks to individual services via the network, through personalized emails with malicious attachments/documents to individuals inside of the network, or using USB sticks as in the case of Stuxnet [7]. Malicious attachments include typical document formats, such as DOC, EXE, JS, PDF, and compressed files (ZIP). The malicious executables in attachments often have interesting properties that distinguish them from normal files. Some have encrypted areas, which often indicate a malware packer. Others import debugger symbols to undermine code analysis. In some cases, writable and executable sections are included, suggesting self-modifying malicious code. Attacks via removable media can contain complex startup routines (cf. [8]). Since the size of the exploit code is often very limited, other code is loaded via the network in a subsequent phase, which often installs a permanent backdoor to *establish foothold*, e.g., by installing remote access routines. The installation of remote access routines usually leaves only little traces (cf. [9]) and the use of exfiltrated access data is hard to distinguish from normal use. However, both acts usually involve an *escalation of privileges* by the attack code which could be detected by host-based intrusion detection systems (see next section). After that, the internal network is being analyzed (*internal reconnaissance*). Usually, it starts with the local analysis of the initially compromised host and its active network connections. In a subsequent *lateral movement* phase more hosts become infected. The detection of hosts in a local area network during the internal reconnaissance typically involves ARP scans, which do not differ from normal ARP requests. An ARP-Scan iterates over all internet protocol addresses of any potentially existing host in a network based on the netmask of the first infected host using ARP requests. Thereafter, in some cases, individual connections between hosts are kidnapped by means of ARP or DHCP spoofing [10]. So far, there are few reliable methods to detect these attack phases. Therefore, the internal reconnaissance and lateral movement phase must be

limited by preventive security measures. In order to ensure a permanent network access (*maintain presence*), different or polymorphic malware families are installed on different hosts. Eventually, the exfiltration of all data interesting for the attacker takes place. The exfiltration of data typically leads to abnormalities in the information flow distribution of network traffic which could be detected by statistical or machine-learning approaches that can differentiate between normal and abnormal behavior.

1.3 Detection Methods and Mitigation Strategies

In order to protect an IT system against attacks *preventive* and *reactive* measures are applied. In this context, in particular intrusion detection systems (reactive measures) are gaining in importance. Network-based IT systems consist of hosts (physical or virtual machines) and intermediate systems (routers, switches). On these grounds it is necessary to distinguish between *host-based* and *network-based* security measures which are discussed in this section.

Host-based Security Measures

A top-down view of preventive host-based security measures starts at the software level because all systems deploy a form of software (firmware, operating system kernel, user space software, applications) for providing their services. To launch an attack on a piece of software, the attacker must know the entry point of a vulnerability to inject malicious code, find out where the code is located in the memory after exploiting the vulnerability, and divert the control flow to the malicious code. Typical injection points are data structures and buffers which can be exploited with *format string vulnerabilities*, *integer overflows*, and *uninitialized variables* (including *use-after-free*[1]). From an attacker's perspective on the central processing unit (CPU) of a system, control flows can be changed by manipulating data structures with reference to procedure calls (e.g., x86 CALL, ENTER, SYSENTER, SYSCALL), software interrupts (e.g., INT), return of control (e.g., RET, LEAVE, SYSEXIT, SYSRET, IRET), jumps (e.g., x86 JMP, Jcc), and by using virtual machine operations on newer processors (e.g., VMLAUNCH, VMRESUME, VMXOFF, VMXON, VMCALL, VMFUNC) [3].

Preventive Host Security

The usability of vulnerabilities and the corresponding control flow manipulation mechanisms for an attack depends on the memory management and the applied (basic) security mechanisms. Figure 1.4 shows the x86 processor memory management, that

[1]Use-after-free: continued use of memory for data structures that have already been released

applies paging and memory segmentation to protect memory accesses against malicious use. The segmentation protects different memory areas (text/code, heap/data, and stack) against accidental overwriting, such as stack overwriting during write accesses to the heap/data segment and write accesses to the text/code segment which contains the current program. Paging is responsible for the mapping of virtual (logical) memory pages to physical memory pages (system random access memory) and adds additional security mechanisms, such as fine-grained read and write permissions on 4KB memory pages (separately for the operating system kernel and user space programs) and the No-eXecute bit in modern processors which prohibits the instruction fetching from the corresponding memory page.

Regarding the memory, one has to distinguish between attacks on the stack segment (typically *stack overflow*) and the heap segment (typically *heap overflow* or *use-after-free*). A stack overflow is a buffer overflow which overwrites the return addresses on the stack – referenced by the *extended base pointer* (EBP) CPU register – to reroute the control flow to an attacker's exploit. Figure 1.5 depicts such a stack segment for the x86 CPU (32 Bit mode). The stack grows from high to low memory addresses. Write accesses to the local variables of a function, however, range from low to high addresses. If a buffer is located below the return pointer in Figure 1.5 (e.g., in the last variable on the stack, referenced by the *extended stack pointer* (ESP) CPU register), and its size is ignored during data transfer, an attacker can overwrite the return pointer and redirect the program control flow to a destination of his/her choice. Targets for control flow rerouting are the buffer itself, fragments from the executed program (return-oriented programming (ROP [11])), and program libraries (e.g., return to libc [12]). On modern systems the stack integrity is typically protected with *stack canaries* [13]. Canaries are random numbers that are placed in the stack directly below the return pointer and above the local variables of a function (cf. Figure 1.5). They are checked when returning to the calling function by compiler instrumented code to detect overflow conditions.

The heap integrity can be protected by means of fine-grained *guard pages* (cf. page frames in Figure 1.4), but no appropriate implementations are known currently. However, the C library on Linux systems contains heap protections for memory management (cf. [14]) which protect the heap chunks using a checksum over the memory address, the chunk size, and a program global random number (canary). There are also protection mechanisms to detect malicious code or code fragments after a successful injection. The main protective mechanism is the *address space layout randomization* (ASLR [15]). Here, the stack and the heap segment are shifted by a random amount in the virtual address space during program start. ASLR was later refined by *position independent code* (PIC/PIE [16]) which shifts the code/text segment by a random value. Figure 1.6 represents the interaction of paging, page-based access rights, coarse granular guard pages, and ASLR on a Linux (x64) host. In a fully protected program the code (code segment) is moved by a random amount from the address 0 (left side of the figure) at program start to obfuscate usable addresses of machine instructions

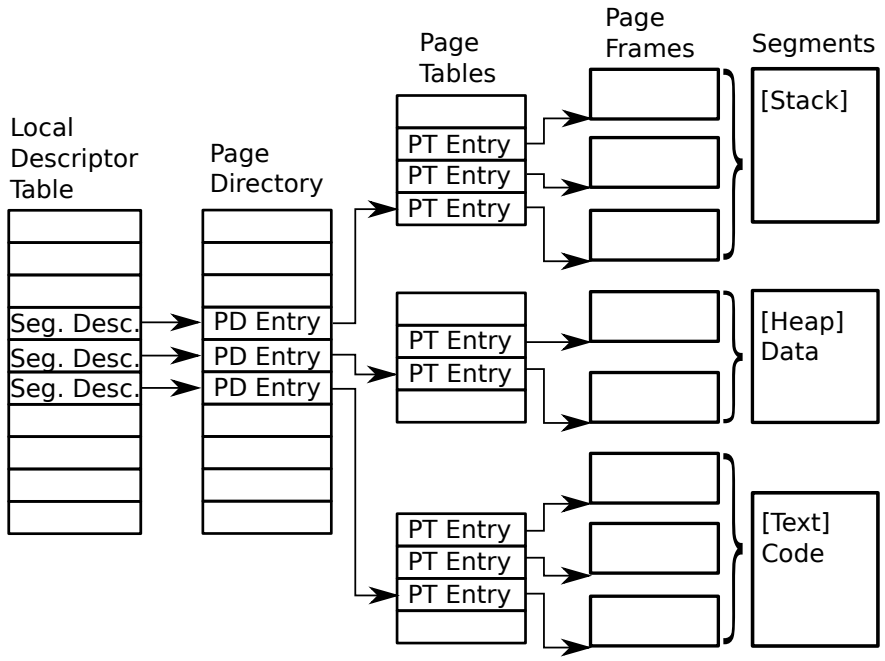

Figure 1.4: Memory paging and segmentation (based on [3])

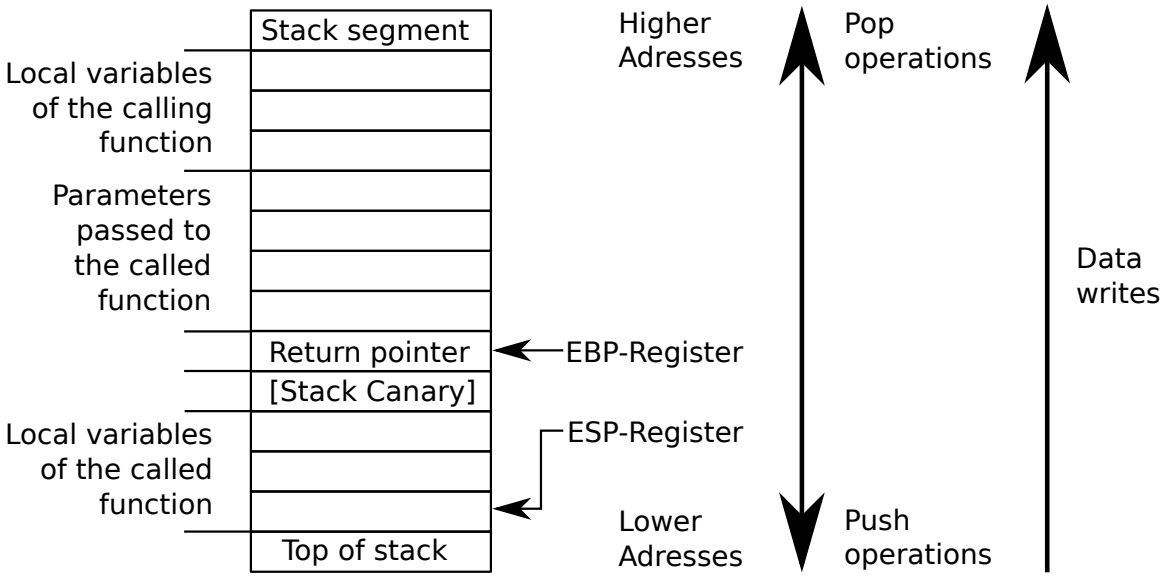

Figure 1.5: Stack segment of a x86 CPU (based on [3])

for an attack. The globally writable variables of the program in the text segment are separated from the heap segment by a non-accessible guard page (marked by dashed lines in the figure). The beginning of the heap segment is moved by a random amount in memory, and shared library code is integrated below the stack segment. Each library is separated through guard pages from the subsequent library. The guard page of the last library separates the shared library section from the stack. Since the stack starts from the highest address, it is also moved down by a random amount in the address space to make it difficult for the attacker to apply stack-based attacks.

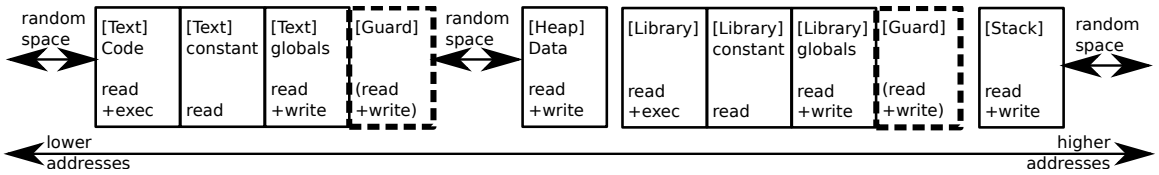

Figure 1.6: Memory layout on a protected Linux (x64) host

More advanced defenses against control flow changes, such as the diversion of return addresses to malicious code, use compiler instrumentation mechanisms to ensure the integrity of the control flow. An example of such a mechanism for stacks is *control flow integrity checking* (CFI [17]). Moreover, all types of code pointers (e.g., function pointers) can also be protected using a mechanism called *code pointer integrity checking* (CPI [18]).

Reactive Host Security

After first attacks were published in the late 1960s [19] and 1970s [20], first reactive systems were developed to detect the malicious use of computing resources. The first approaches addressed only host-based measures. The involved components were mainframes, terminals, and tape drives. Conceptually, attack detection comprises sensors that gather *security-critical events* and an expert system evaluating these events subsequently. Accordingly, an initial distinction was made in the first attack detection design [21] between customer audit trails and *security audit trails*. The former served exclusively for accounting purposes, while the latter covered security-critical events. The first observed security-critical events were repeated logins, duration and number of login sessions, job start time, job run-time, the number of jobs, the frequency of data access, the frequency of program access, the number of characters transmitted to a terminal, and the number of records written as program (e.g., compilation) or data to determine: (1) the use of systems outside of normal daytime, (2) abnormal frequency of use, (3) abnormal data access (mostly volume), and (4) abnormal *patterns* of program or data access [21]. Network analysis in the sense of an audit of all network components had been also considered, but it was too much effort for the audit data analysis at that time. The analysis time was decoupled from the occurrence of the

events (job running the expert system, once a day). It was intended to log audit trails on tape for later analysis, what in fact represents a precursor of computer forensics.

Based on the first IDS concept, a first host-IDS model was published in the late 1980s by Dorothy Denning [22] followed by *real-time* implementations of this model, e.g., IDES [23] and HAYSTACK [24]. The name of the first implementation – *Intrusion Detection Expert System* (IDES) – has introduced the term of the *Intrusion Detection System* (IDS). Real-time capability was achieved by decoupling the audit data generation and the analysis which was performed on a separate system. The analysis model consisted of the following components. *Subjects* (normally users) are the initiators of activities on the target system. *Objects* are resources which are managed by the system, e.g., files (read and write access), commands (execute activity), or devices (e.g., access to printers). *Audit trails* capture the activities of subjects on objects. *Activity profiles* cover audit trails by means of *statistical metrics* that are matched against a *statistical model*. In this case, the metrics are *event counters* that count the number of specific events in an audit trail or *interval timers*, which capture the time interval between two events. *Operational models*, *mean and standard deviation models*, *multivariate models*, *Markov-process models,* and *time series models* have been discussed as statistical models. Operational models assume that abnormalities can be detected by comparing the metrics with a fixed threshold, e.g., the number of failed password attempts. Mean and standard deviation models assume that a new observation can be defined to be abnormal if it falls outside of a confidence interval, i.e., a certain standard deviation from the mean of previously observed events. A multivariate model attempts to uncover whether combinations of multiple events correlate with intrusions, e.g., IO load with CPU load. Markov process models assume that a new observation can be defined to be abnormal if its probability as determined by its previous state (previously observed event) and a transition matrix, which characterizes transition frequencies between observed events, is too low. Time series models use interval timers together with event counters to measure inter-arrival times between observed events. A new observation is abnormal if its probability of occurring inside of the observed interval is too low. These models had been representing the basis for host-based IDS for many years.

Network-based Attacks and Security Measures

Attacks from the network require a further glance on the communication layers involved. Networks are essentially described through the communication protocols applied among the networked systems. A distributed application (application layer) needs auxiliary protocols for a stable end-to-end data transport (transport layer), the reachability of all intermediate nodes on the path to the target system (routers on the network layer) and the access to local links to reach the first intermediate devices (switches on the link layer). Web applications often implement additional protocols

using the application layer only for data transport (e.g., XML-RPC [25], SOAP [26]). These protocols can be regarded as an overlay layer in terms of the protocol stack. Figure 1.7 shows such a protocol stack. The exploitability of the individual layers and protocols for attacks depends on the location of the attacker.

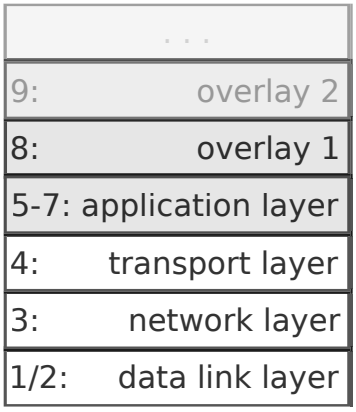

Figure 1.7: Protocol stack

If the attacker is located in a local area network (LAN), he/she can hijack connections by manipulating the auxiliary protocols of the data link and network layers. In particular, the auxiliary protocols of the Internet Protocol version 4 (IPv4 [27]), namely the Address Resolution Protocol (ARP [28]), the Dynamic Host Configuration Protocol (DHCP [29]), and the Internet Control Message Protocol (ICMP [30]), and IPv6 [31] in conjunction with ICMPv6 [32] and DHCPv6 [33] are vulnerable to attacks of this nature (see Section 2.1). If the attacker is located outside of the local area network he/she has to rely on attacks on the routing protocols (e.g., Border Gateway Protocol [34]), on the transport layer (e.g., sequence number prediction for the Transmission Control Protocol [35]), and on the application layer (e.g., attacks against the domain name system such as Dan Kaminsky's DNS attack[2]). Attacks on the application layer target more frequently on the incorrect use of the transmitted data structures in combination with a specific memory model, as already discussed above (cf. Fig. 1.4 and Fig. 1.5), rather than a specific protocol.

Preventive Network Security

In the area of network-based security technologies, significantly less research has been performed compared to the host-based area. Figure 1.8 represents a typical network security setup as it is installed for private users and small to medium-sized companies. The basic components of such a security setup are a *firewall* with a supplementary *application-level gateway* (ALG) and an *intrusion prevention/detection system* (IPS/IDS). Typically, the two components are combined in an intermediate system (router). The firewall functionality can range from a simple packet filtering, which processes the layers 1 to 3 (see Figure 1.7), to an application-level gateway which additionally analyzes the application layer. Historically, ALGs were responsible for transparent application-specific address/port translation between private and public IP addresses of the network [36]. To realize this functionality they had to rewrite the application payload. The ability to rewrite network protocols and application payload made them interesting for security applications. The interfaces between the ALG and the firewall are used today on the one hand to check whether a transmitted application protocol adheres to its specification, and on the other hand to rewrite application

[2] `https://dankaminsky.com/2008/07/24/details/`

data with the purpose to *normalize* potentially malicious content by transferring it into a harmless representation. A closer look at the details of the figure reveals an implicit assumption about attacks at the network level. The positioning of the firewall in relation to the IDS implies that the network to be protected is attacked only from outside because the firewall/ALG usually must prevent *analysis circumvention/evasion* attempts. Analysis circumvention methods range from the fragmentation of the Internet Protocol (IP) [37] to fragmented or otherwise malformed TCP and higher layer packets [38]. Network devices with full ALG/IDS capabilities prevent such attempts, e.g., by dropping fragmented IP packets, and by observing the connection state of TCP or by normalizing the TCP packets [39, 40].

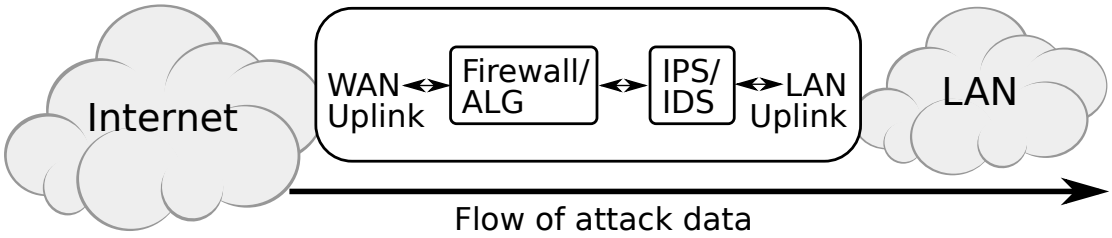

Figure 1.8: Network security setup

Reactive Network Security

Parallel to the development of host-IDSs, the first attacks on networks were published, e.g., [41, 42]. These publications led to the design of the first *network security monitor* (NSM) [43, 44]. The NSM analyzes the network traffic by means of a four-dimensional matrix (similar to an access control matrix) which has the following axes: *source* (sending host), *destination* (receiving host), *service* (mail, rlogin, whois, NFS, ...), and a unique *connection identifier* for each connection. In addition to this matrix, there were three other matrices, each with a reduced dimensionality. The *source-destination-service* matrix aggregated all traffic of a source to a destination with a specific service, the *source-destination* matrix aggregated all traffic across all services between a source and a destination, and the *source* matrix aggregated all traffic of a source. As a first preliminary attack detection measure, rule-based analyzes were implemented to provide an insight into the network traffic. The source matrix was used to determine hosts that connect to more than 15 other hosts or non-existent hosts. The source-destination-service matrix was used to find services that are requested (according to the implementation description within 5 minutes) more than 15 times from the same host. With these rules, a real attack from the outside, an internally stimulated attack for analysis purposes, and unknown communication protocols were detected. In addition, it was planned to use the IDES model (cf. Reactive Host Security) to detect anomalies in the network traffic, which, however, was not implemented. Instead, the *network security monitor* was combined in subsequent work with the HAYSTACK host-IDS to form the first distributed IDS (DIDS [45]) which correlated events from the NSM

and HAYSTACK to assign a network (user) identity to each connection and to track objects moving around the network. These first prototypes performed all analyses only on the network and transport protocol layer. An analysis of the application protocol layer was ruled out as impractical.

During the late 1990s many commercial IDSs were offered to the market. Commercial network intrusion detection functionality starts with simple filters for *denial-of-service* (DOS) attacks on the packet level of layers 2 and 3, e.g., Ping of Death [46], LAND [47], SMURF [48], or UDP Port Loopback attacks [49, 50], or stealth network scans, such as the TCP Null Scan [51]. This type of IDS is typically applied to private consumer routers and its integration goes back to the early days of private internet access, when implementation weaknesses in the stacks of the ISP customers were often crashed by malicious internet users. More powerful IDS solutions include full network-based IDS (NIDS), such as SNORT [52], which is used on routers in larger companies. The main difference from the IDS on small devices is the application of complex attack analyses which require an examination of the application layer and the full packet payload (Deep Packet Inspection (DPI)). SNORT is currently the most prominent full-featured NIDS because its source as well as the *signatures* of known attacks were published from the outset and constantly updated by its designer.

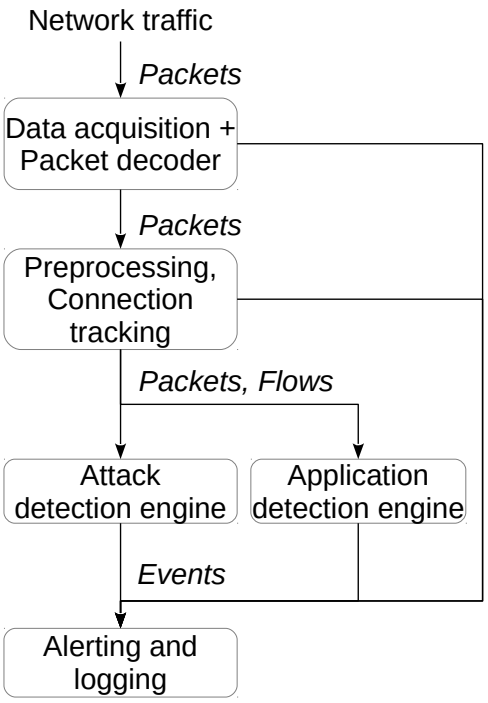

Figure 1.9: Architecture of SNORT

SNORT consists of four subsystems (cf. Fig. 1.9): the *packet decoder*, the *connection tracking*, the *detection engine*, and the *alerting and logging* subsystem. The packet decoder maps the data structures and protocol fields of raw network packets according to the TCP/IP network stack into an internal representation. It has a lower layer, which ensures that the different packet types of the link layer, network layer, and transport layer are built according to their specification and have no wrong size information or unusual entries in the protocol fields. Thus, it can also identify the aforementioned packet-based DOS attacks. The connection tracking preprocesses decoded data depending on the identified protocol. Its analysis capabilities are beyond the capability of simple packet filters. Fragmented IP packets are defragmented prior to further processing and TCP streams are assembled from several individual IP packets. Depending on the packet type, it may also be necessary to normalize packets. In the application layer, it is, for instance, necessary to provide a unique representation of *uniform resource identifiers* (URIs) for HTTP packets. For this purpose,

hexadecimal or unicode characters are translated to a common notation and other obfuscation attempts of attackers in the URI addressing are corrected, e.g., relative URI path traversals. The normalized data of the application layer are evaluated by a complex signature analysis in the *detection engine* which applies IDS rules to packets or streams. A SNORT signature for an attack comprises a header and optional conditions. The header specifies the basic parameters of the signature, such as the inspected protocol (IP, UDP, TCP, or ICMP), the observed communication directions (uni- or bi-directional), the source and destination IP addresses, and the port numbers. Optionally, actions may be specified that are executed when the rule matches, e.g., generating alarms or dropping datagrams. The option part specifies constraints for matching header and payload fields of datagrams. Protocol header constraints are used to match datagram header fields. The considered fields depend on the protocol selected in the rule header, e.g., sequence number fields can only be matched for TCP segments. Payload constraints can be used to match values or patterns, e.g., strings, in a datagram or transport stream. If SNORT identifies a datagram or transport stream that matches all constraints of a rule header and all of the protocol header and payload constraints, the action defined in the rule header will be executed. A more detailed analysis of the signature language of SNORT has been published in [53].

1.4 Attack Severity Rankings

Impacts of attacks are ranked generally in accordance with the attack phases (cf. Section 1.2). The IDS SNORT, for example, divides the severity of attacks into four priority classes. The fourth and least significant priority class contains only one event: *"A TCP connection was detected"* which corresponds to no attack. The third priority class summarizes events of the initial reconnaissance phase. Events of this class range from the detection of network scans, via ICMP activities, and connections to potentially vulnerable services, such as desktop sharing and network management systems, to the detection of suspicious strings, such as the identification of exploit kit banners. The second priority class summarizes the phases of internal reconnaissance, establishment of foothold, lateral movement, and exfiltration of data. The events of this class range from the detection of information leaks of services, suspicious logins, remote procedure calls, unusual client port connections, use of manipulated protocol data units, use of vulnerable web applications, and transfer of suspicious filenames to the use of suspicious services. The first and most critical priority class summarizes all the events of the initial compromise and escalation of privileges. Events of this class are the detection of transmitted shell code, attacks on web applications, activity of known trojans, expansion of user privileges, and the use of administration privileges.

1.5 Open Research Challenges

Capturing traffic to detect attacks on larger administrative network domains, e.g., an enterprise network composed of multiple subnets, is nowadays typically centralized by picking up and analyzing traffic on the uplink to the internet. This approach allows to identify attacks from the internet, but it has though a number of important disadvantages. *Insider attacks* are not detected regardless of whether they are initiated deliberately or triggered by compromised devices. External attacks are equally difficult to detect because the initial compromise often takes place via mail by means of unknown vulnerabilities in file attachments or by simple social engineering, which tricks the recipient to run executable code from mail attachments. In addition, threats, such as references to external web-based content as they are used for phishing and attacks using web-based content, are not recognized by existing preventive and reactive security measures.

Figure 1.10 illustrates an example of the issues with current monitoring technologies. (1) The *initial compromise* is carried out by means of a contaminated USB stick on a PC in a local area network (LAN). As these activities take place only locally on the PC, they are outside the viewing range of network-based monitoring systems. (2) In order to attack the servers in another LAN assumed to contain data of interest for the attacker *an escalation of privileges* is performed to bridge the router between the two network segments. This can be done, for instance, using a domain-controller-based login on a PC in the respective LAN with locally captured login information from the first PC – a step which is difficult to detect because the login may be legal. (3) Thereafter the attack/access program needs information about the upper LAN segment. For this purpose, scans of the link layer are used in some cases to determine other systems (*internal reconnaissance*). Existing monitoring methods, e.g., the flow analysis, are based primarily on accounting information of the network and transport layer [54, 55] and are thus unable to detect actions on the data link layer. Therefore, an attacker can use any link layer attack to propagate in the upper LAN segment (*lateral movement*) to collect more data from all servers and to move the data within the network segment. Moreover, the attacks do not have to be limited to the data link layer. Due to the slow implementation of the next internet protocol standard (network layer), IPv6-based attacks are often also overlooked by current monitoring systems. Chapter 2 discusses a number of link layer attacks for IPv4 and IPv6 that can be typically observed in such a scenario. (3') The problem is getting worse if an attacker encounters a *virtual machine* (VM) host. The attacker can spread freely inside the virtual machines, since this area is not covered by current monitoring technologies. (4) Finally, the collected data must be moved out of the network (*exfiltration of data*). According to analyses of targeted attacks, such as the Regin framework [56, 57], this step can also be performed quite stealthy, e.g., by means of the server message block (SMB) [58] protocol for intermediate stations (data link layer variant) and a Transport Layer Security (TLS) socket [59] for the final move out of the network.

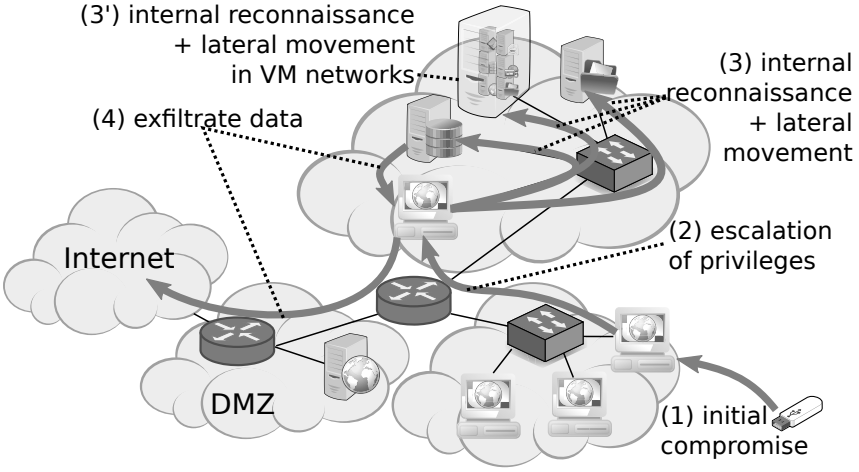

Figure 1.10: Example scenario of an internal/targeted attack

For solving some of the issues with insider and targeted attacks the security measures have to be moved into the corporate or private network and also recursively into the VM networks. This brings some challenges. (1) A preventive security concept for the internet uplink, which consists of packet filtering, proxies, and an application-level gateway (ALG), is easy to implement, as there is only one data path, but what about a concept that ensures an equivalent security for a local area network or subnet consisting of network switches with multiple ports and multiple data paths? (2) Subsequently, if the reactive mechanisms (e.g, IDSs) are moved into the local area network, a further problem arises in connection with the massive data throughput of local area networks compared to the limited throughput on the uplink. The throughput of local subnets cannot be handled by existing IDSs.

Challenge 1 – Preventive Measures for Local Area and Virtual Machine Networks. The solution of the first challenge requires a centralization of the network management that goes beyond existing concepts for remote maintenance, such as the Simple Network Management Protocol (SNMP [60, 61]). One possible approach is to centralize previously decentralized network services, e.g., routing and auxiliary protocols for address resolution which are managed directly by the network devices at present, into software-based controllers with a general overview of the domain.

In the 1980s, first network services were centralized in *stored program controlled* (SPC) telephone networks [62]. For this purpose, switch software was installed in the network on SPC-enabled hardware which could configure the forwarding devices (virtual-circuit telephone switches) to use service primitives, e.g., call routing, the creation of billing records, automatic announcements, and the collection of information (numerals) through prompts [63, 64]. In the 1990s, the idea was further developed within the *active networks* [65] that combined different concepts. In their simplest form var-

ious network primitives, e.g., the forwarding of a packet within a switch to an output port, were remotely controlled by an out-of-band communication channel. However, there were also radical ideas that aimed at programmable network devices via in-band communication, e.g., using active network packets containing bytecode for a network program and payload data. In 2004, the former idea was taken up again for data-based wide area networks. The SoftRouter model of the Bell Labs [66] envisaged to limit routers to their basic functionality as *forwarding elements* (switching of packets based on the longest prefix match of the IP addresses), whereas the routing protocols themselves should be implemented in general-purpose computers (*control elements*) that communicate with the forwarding elements using standardized protocols. A possible standardized protocol between the control and forwarding elements should emerge from the development of a parallel ongoing effort called *forwarding and control element separation* (ForCES) framework [67]. The two approaches mentioned security measures as a potential advantage of software-based network management – an idea that has never been implemented for wide-area networks.

The *Ethane* software switch [68] for local area networks followed a similar concept as the SoftRouter, but with a strong focus on improving security this time. Ethane couples flow-based Ethernet switches with a centralized controller that knows the global network topology and grants access by explicitly enabling permitted flows within the network switches along a centrally computed route. The controller enforces a strong binding between a packet and its source by restricting the port access to a switch on the IP addresses assigned via DHCP. One of the biggest problems that were reported in this implementation is the handling of broadcast traffic. Most broadcast traffic is caused by address resolution protocols, e.g., ARP, which generate a huge load on the controller. Other address resolutions, such as the IPv6 neighbor discovery, had apparently not been implemented resulting in further shortcomings with regard to the spoofing of Ethernet/internet addresses. Accordingly, there is a need for research on a method that simultaneously *limits broadcast traffic* and *implements the address assignment and address resolution in a secure manner* for all major protocols (IPv4 + IPv6 including auxiliary protocols). The need for a standard protocol between the controller and the switch was noted also by this approach. The corresponding protocol was published by the authors of Ethane under the name *OpenFlow* [69]. Unfortunately, it was never tried to reproduce the Ethane approach based on the OpenFlow protocol and to solve the mentioned problems. Subsequent publications were limited to the analysis of the control channel and the application security at the controller level [70, 71]. They have less considered the possibilities for an increased network security using OpenFlow or ported only classical firewall mechanisms and monitoring procedures [72].

Challenge 2 – High-speed Deep Packet Inspection for Local Area Networks. A major challenge for the monitoring of local area networks are the rapidly increas-

ing data rates which often lead to an uncontrolled discarding of traffic in overload situations in the monitoring stations. For DPI-based intrusion detection systems, numerous approaches to improve the throughput have been proposed. They range from hardware-based solutions [73, 74] via the parallelization of analyzes [74, 75] to the use of more efficient analysis algorithms [76, 77]. Unfortunately, most hardware and parallelization approaches switch off essential intrusion detection functions for evaluation, such as preprocessing (e.g., reassembly of TCP streams), rule evaluation, and logging. Thus, the analysis is de facto deactivated, i.e., these configurations are not able to detect any real intrusions.

To avoid the bottlenecks of a centralized analysis, distributed approaches have been examined for a long time. The approaches range from systems that perform a central data analysis and event correlation [45, 78] via hierarchical structures, which distribute the analyses and event correlations across different processing levels, up to fully distributed multi-agent paradigms [79, 80] and fully decentralized peer-to-peer (P2P) paradigms [81, 82]. A closer look on the proposed approaches shows that many of them are limited to the collection, distribution, and aggregation of information about observed suspicious activities [82], to adapt the defenses to the acquired knowledge [79, 80]. Another important aspect of distributed approaches is to establish a cooperative security to improve the analysis accuracy [83]. However, there are rarely approaches that leverage the distribution to reduce the burden of the analysis [81], e.g., by transferring the analysis in overload situations to another system or by discarding non-relevant traffic. In summary, research approaches that parallelize intrusion detection systems with all of their components under real-life conditions are still missing.

Special Issues. Further special issues in the analysis of external attacks particularly go back to the massive use of web technologies in today's networks. The development of solutions for web-based attacks requires new preventive and reactive security measures. As already discussed in Section 1.3, the transmitted application data (potentially malicious interpreted scripts) are typically deeply buried from a network perspective in the application layer. The current research on the protection of web applications includes server-side, client-side, and hybrid approaches. Server-side web security approaches usually require modifications of the web applications [84, 85, 86] to distinguish between trusted application data and untrusted external data. They require sometimes mandatory support from the client-side browsers [87, 88] to enforce a server-specified policy at the client. Client-side approaches comprise systems to detect information extraction (cookies) and aggressive scripts (e.g., prevention of window closing) [89]. Sometimes dynamic data tainting is added to track the indirect usage of sensitive data sources, e.g., the aforementioned cookies [90, 91]. Furthermore, anomaly-based detection systems create a profile of the application usage of JavaScript and enforce it later [92]. In addition, there are signature-based systems which remove

authentication information from requests [93] and systems based on machine-learning which can detect JavaScript-based malware [94].

A main issue for the analysis of web documents is that increasingly so-called overlay structures are used, i.e., multiple protocols are nested within each other at the application layer [95]. Web 2.0 technologies are an example for this. Current commercial application layer proxies and intrusion detection systems inspect the traffic at least up to the HTTP layer and thus provide the possibility of classifying and filtering the traffic based on *Uniform Resource Locators* (URLs). Some of them also scan traffic for viruses or other (binary) malware but most of them are not capable of inspecting the traffic for script-based attacks, to normalize HTML to a certain save representation level, or to remove malicious scripts and other potentially malicious object embeddings. New firewall/NIDS concepts are urgently required for a semantic analysis of multiple nested application protocols.

1.6 Structure of This Thesis

The structure of this thesis follows the attacks against different layers of the network protocol stack and the resulting challenges discussed previously. This is illustrated in Figure 1.11 through a network stack that extends the classical TCP/IP stack upwards and downwards. The lowermost part of the network stack expands the TCP/IP stack to metadata regarding switch-internal data related to port-to-MAC mappings and other data, which are necessary to detect attacks on network components, such as switches and VM bridges. The next four layers (upper part of link layer – lower part of application layer) are already largely covered by traditional network IDS methods. The expansion of the network stack above the application layer (gray area) relates among others to modern Web 2.0 communication paradigms that use the application layer (e.g., HTTP, web sockets) only as an additional transport layer for nested protocols (e.g., SOAP). This part requires specialized application-specific IDSs to still be able to detect attacks, since traditional network IDS methods assume a comparatively flat IP stack.

The first part of this work covers the white area of the stack. It begins with Chapter 2 which examines various security measures to prevent internal attacks on local and VM networks that are directed against the network-connecting elements and the synchronization of the layer 2 and layer 3. The chapter addresses challenge 1 with a software-defined networking (SDN) approach to prevent the investigated attacks. Chapter 3 proposes measures to the acceleration of the traditional NIDS analysis in order to establish a high-speed monitoring for local area networks. This addresses challenge 2 with a parallelization concept.

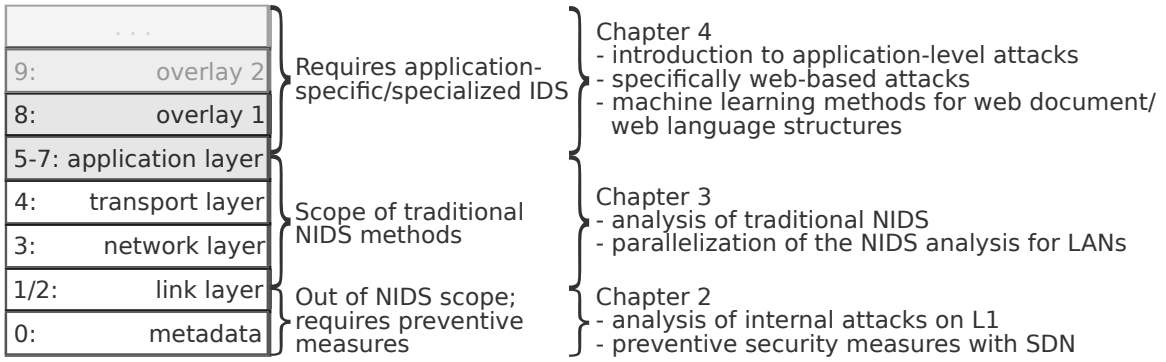

Figure 1.11: Structure of this thesis in relation to different parts of the network protocol stack

The second part in Chapter 4 of this work covers the gray area of the stack. It discusses specialized NIDS methodologies for a network-based analysis of web protocols and applications. The first part of the chapter describes typical cross-site-scripting (XSS) attacks and discusses the problems of parsing web languages in perimeter firewalls and other security solutions. The second part of the chapter presents a firewall architecture which applies novel NIDS methods based on machine learning to identify web applications and to ward off malicious inputs.

2 Restriction of Internal and Targeted Attacks

The detection of attacks on network domains is nowadays usually accomplished centrally by analyzing the data traffic on the uplink to the Internet. The first phase of an infection of an advanced targeted attack (phase of initial compromise) is usually difficult to control. Often the attackers use external media, such as USB sticks, hardware with preinstalled malware, or contaminated mobile devices to infect target systems. In such scenarios, the initial infection cannot be blocked at the network level. The lateral movement of attack programs (exploits) through internal networks and the exfiltration of data, however, which are the real purpose of targeted attacks, run always over the network. Security measures against internal network attacks require a comprehensive sensor system that spans the entire network to the network perimeter. Especially for preventive measures, this means providing a security concept for local area networks. This chapter discusses, based on an analysis of past LAN-based attacks, a possible solution for this problem. As part of the solution, Software-Defined Networking (SDN) [96] is applied as a vehicle for centralizing information on all network activities in a central authority – the SDN controller – that manages all network connections and hence the associated data flows. As a side effect of using SDN, networks of virtual machines on a single host, which represent a blind spot for network monitoring so far, can also be integrated into the defensive measures.

2.1 Classical Threats for Local Area Networks

In Ethernet networks there are plenty of vulnerabilities that allow a traffic redirection with the possibility of reading and overwriting of content. They attack the layers 2 and 3 of the TCP/IP stack. These vulnerabilities exist for both physical and virtualized networks. For physical systems due to their continuous development, there are meanwhile solutions that protect against some of these attacks to a certain extent. In virtual machine hosts, however, man-in-the-middle attacks are still possible between guest systems without much effort. The Linux bridge used by default does not provide protection against these threats. Next an analysis of these attacks that are applied in the code of freely-available tools – such as Ettercap[1] – is presented.

[1] https://ettercap.github.io/ettercap/

Internal Reconnaissance – Network Scan

In a first step an internal attacker or attack program has to explore the local area network. First information can already be obtained on the internal/compromised hosts. The current IP address, the subnet mask and/or the default gateway address may reveal the maximum size of the network because gateway addresses are usually reserved to the upper end of the network range. Then the attacker can scan the network range based on the previously acquired information. The main scan variant used is the Address Resolution Protocol (ARP [28]) scan.

ARP-Scan. The purpose of the ARP scan is to look for active devices in the subnet (see Figure 2.1). (1–4) For this, the attacker (e.g., station A in Figure 2.1) usually generates a list of all possible host addresses and checks them using ARP requests (*ares_ op$REQUEST*). The addresses are shuffled to request them (*ar$tpa*) in random order. (5) If there is a host that matches one of the addresses it responds with an ARP reply (*ares_ op$REPLY*), containing its hardware address (ar$sha = MAC(<Host>)). To perform this scan the attacker requires a significant large number of requests (#Requests >> #Hosts).

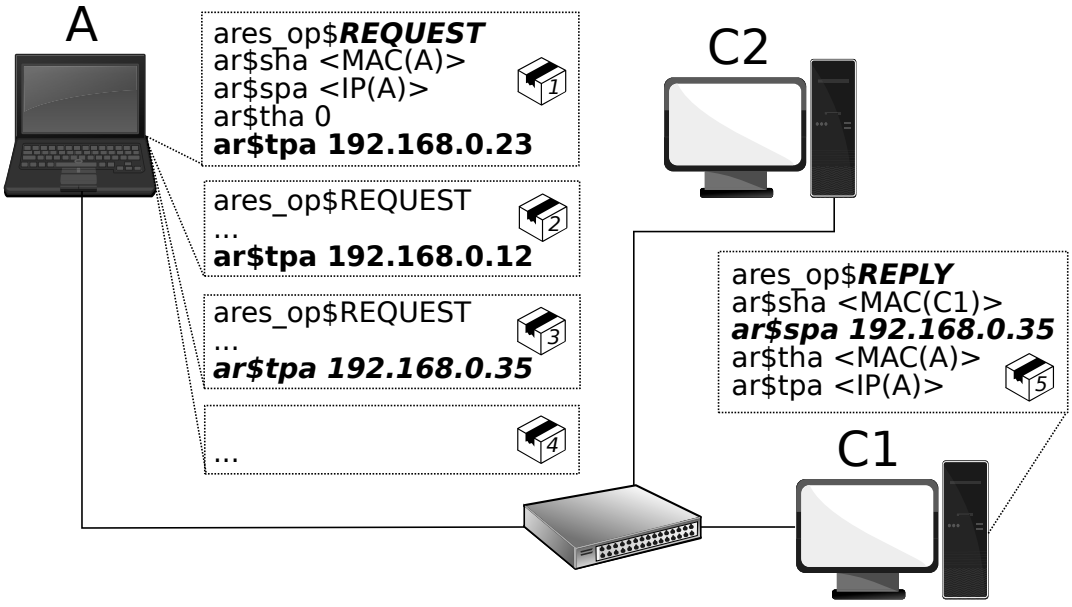

Figure 2.1: ARP Scan

Internal Reconnaissance – Man-in-the-Middle Attacks

After determining potential targets of interest, e.g., routers or servers, in the previous step, the attacker is capable of kidnapping individual or even all links in the network in a further step. In this respect, one has to distinguish between attacks with *half-* and *full-duplex* capabilities. Half-duplex attacks kidnap only one-way communication (e.g., to the internet), e.g., by spoofing the hardware address of the gateway. Data that are routed from the client through the gateway to the internet may be intercepted and manipulated by the attacker. The responses in the reverse direction, however, are sent directly to the client. Full-duplex attacks can manipulate communications in both directions. The three most effective ones are (with ascending complexity) ARP spoofing, port stealing, and DHCPv4 spoofing combined with DNS spoofing.

ARP Spoofing. ARP spoofing aims at associating the attacker's MAC address with the IP address of another host to redirect traffic for this IP address to the attacker's host. There are two variants of this attack: request spoofing (see Figure 2.2) and response spoofing. (1–2) The attacker sends an ARP request with its own MAC address (*ar$sha*) and the IP address to be hijacked (*ar$spa*) via broadcast to the LAN. All hosts update their ARP caches. The subsequent communication to the hijacked IP address goes directly to the attacker. If the attacker wants to hijack the communication between two systems (e.g., *C1* and *C2*), he/she must send each an ARP packet for each IP address to control their communication. The alternative response spoofing variant uses ARP reply (*ares_op$REPLY*) which is sent directly to the two hosts (ar$tha = MAC(C1)||MAC(C2), ar$tpa = IP(C1)||IP(C2)) with the same information for *ar$sha* and *ar$spa*.

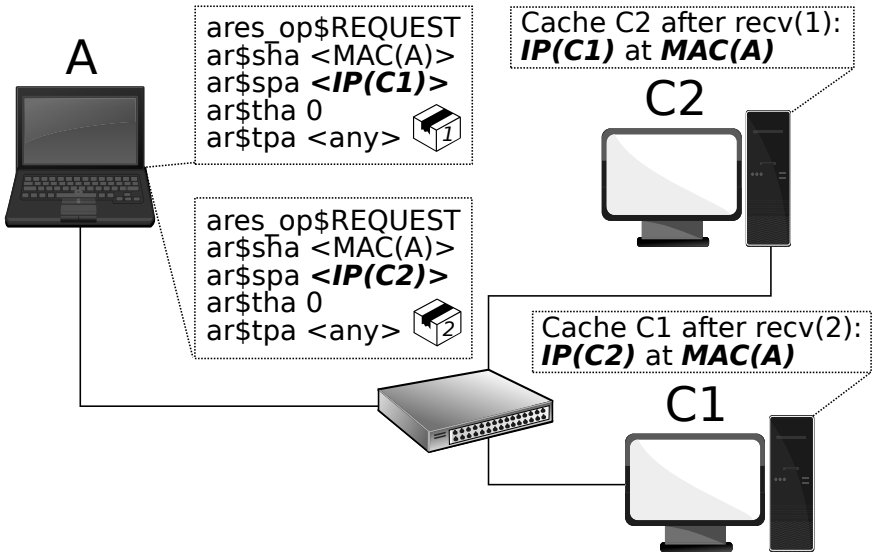

Figure 2.2: ARP-Spoofing

Port Stealing. The purpose of port stealing is to "steal" traffic that is directed to another port of an Ethernet switch (see Figure 2.3). If an attacker A wants to directly take over packets addressed to another host *C1* from the switch he/she first must delete the port registration of *C1*. (1–3) This can be achieved by repeatedly sending ARP packets to the switch in which the the source hardware address is the one of *C1* and the destination address is the hardware address of A. The switch assigns the hardware address of *C1* to the port of A, but it does not forward the ARP packets because A has addressed itself, i.e., the attack is stealthy. (4) If the attacker receives a packet to *C1* (in the example from *C2*), (5) he/she sends an ARP request via broadcast to *C1* and asks for its address. (6) After receiving the ARP reply, the attacker knows that the switch has registered the address of *C1* again to the original port and can forward the intercepted (and possibly manipulated) packet to *C1*.

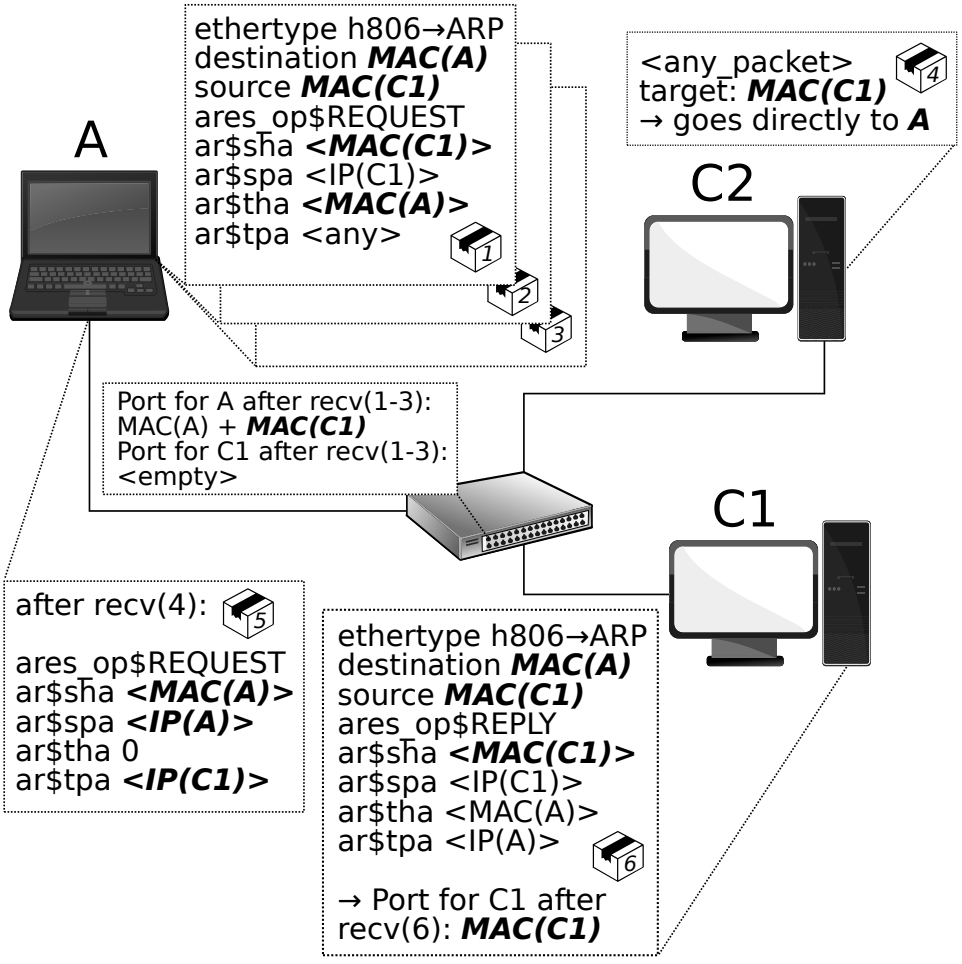

Figure 2.3: Port-Stealing

DHCPv4 with DNS Spoofing. Dynamic Host Configuration Protocol [29] spoofing aims at allocating wrong configuration parameters to the requesting host, e.g., a compromised DNS server, to selectively redirect its data traffic to hosts controlled by the attacker. If an attacker wants to bi-directionally intercept packets using this attack he/she must perform a combined DHCP and DNS spoofing. All necessary information can be transmitted by means of DHCP options [97]. Figure 2.4 illustrates the procedure of the DHCP spoofing. (1) If the attacker receives a DHCP DISCOVER broadcast, (2) he/she sends a DHCP OFFER with an IP address under attacker control as DNS option. The client receives competing offers – the attacker's offer and (3) the offers from regular DHCP servers. Usually the choice falls on the first received offer. The real DHCP server has no chance to win this race because it has to check in its database according to the RFC standard (a) whether the requested address is already in use (possibly with an additional ICMP Echo Request) and (b) it must reserve the address temporarily before sending the offer. (4) After selecting an offer the client officially requests the configuration parameters via broadcast (DHCP REQUEST) which implicitly declines the other offers (*C2* in Figure 2.4). (Not shown) The attacker confirms the selection (DHCP ACK). When the client contacts the attacker's IP address for DNS queries, the attacker can selectively redirect traffic for certain domains to its computer using fake DNS answers.

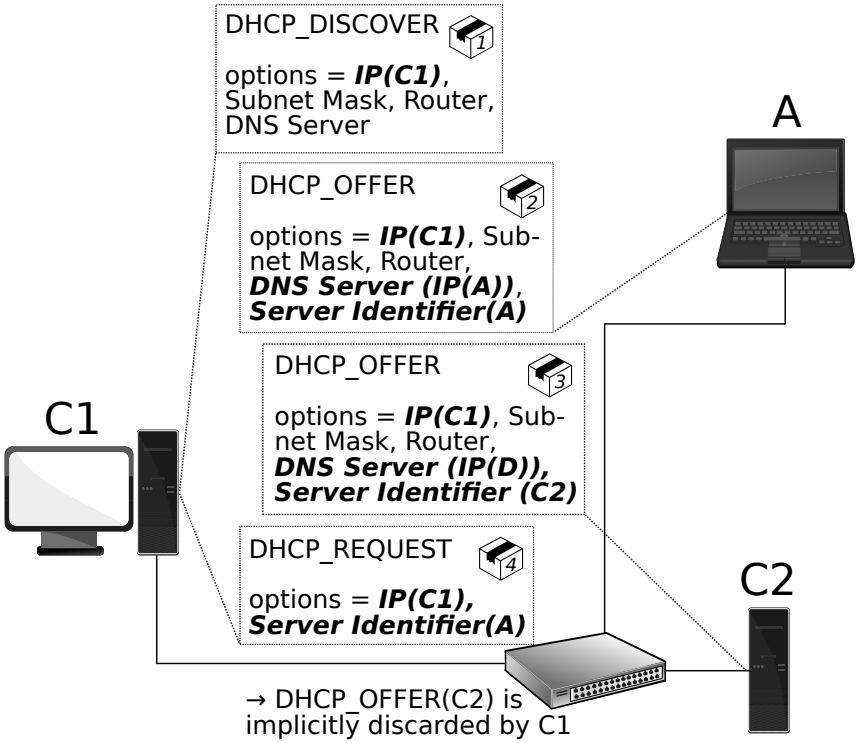

Figure 2.4: DHCPv4 Spoofing

2.2 IPv6-based Attacks

The attacks in the previous section exclusively aim at IPv4. Due to the slow spreading of IPv6, however, a situation has arisen in which each installed network device (router, host) is IPv6-capable, while the protocol is not used actively. Due to the intended transition from IPv4 to IPv6, the IPv6 protocol has automatically priority in the case of a simultaneous configuration of IPv4 and IPv6 parameters. Attackers can use this fact to examine the network using IPv6 methods and to hijack individual connections with its auxiliary protocols. Existing monitoring methods are often not able to analyze the IPv6 protocol – a situation that makes IPv6 attacks particularly attractive. This section presents an IPv6 network scan and two methods to hijack connections which are based on an analysis of *the hacker's choice IPv6 attack toolkit* (THC-IPv6)[2].

IPv6 Multicast Alive Scan. IPv6 does not support ARP to determine the allocation of an IP address to a MAC address. In IPv6, active addresses can be determined through a network discovery using multicast alive scans (see Figure 2.5). (1) The attacker sends only a single ICMPv6 EchoRequest packet [98] with an invalid IPv6 destination option [31] to the all-nodes multicast address (FF02::1). If the attacker is only interested in local routers he/she can choose the all-routers multicast address (FF02::2). (2–3) All nodes in the network reply with an error message (ICMPv6 parameter problem) due to the incorrect option which contains their address in the IPv6 header. A similar network scanning is possible using multicast listener general queries [99] to the address FF02::1. In this case, the hosts respond with multicast listener reports for each network interface. An advantage of the second approach is that the packet rate of the responding hosts to the multicast queries is lowered by random delays in the response, which enables an evasion of anomaly detection systems.

ICMPv6 Neighbor Discovery Spoofing. IPv6 neighbor discovery spoofing follows the same procedure as ARP spoofing, but it uses the ICMPv6 neighbor discovery protocol [100] for this purpose. Unlike ARP, the ICMPv6 packet (ICMPv6 type = 136) contains a special override flag which enforces the overwriting of the cache entry even if it already contains another information for the respective host.

ICMPv6 Router Advertisement, DHCPv6, and DNS Spoofing. The basic idea behind ICMPv6 router advertisement spoofing [101] is the same as with DHCPv4 and DNS spoofing: to assign a DNS server to all local hosts that is under attacker's control. Figure 2.6 depicts the procedure for the IPv6 variant of DHCP and DNS spoofing. (1–3) In the first step, the attacker sends a fake router advertisment with

[2] https://github.com/vanhauser-thc/thc-ipv6

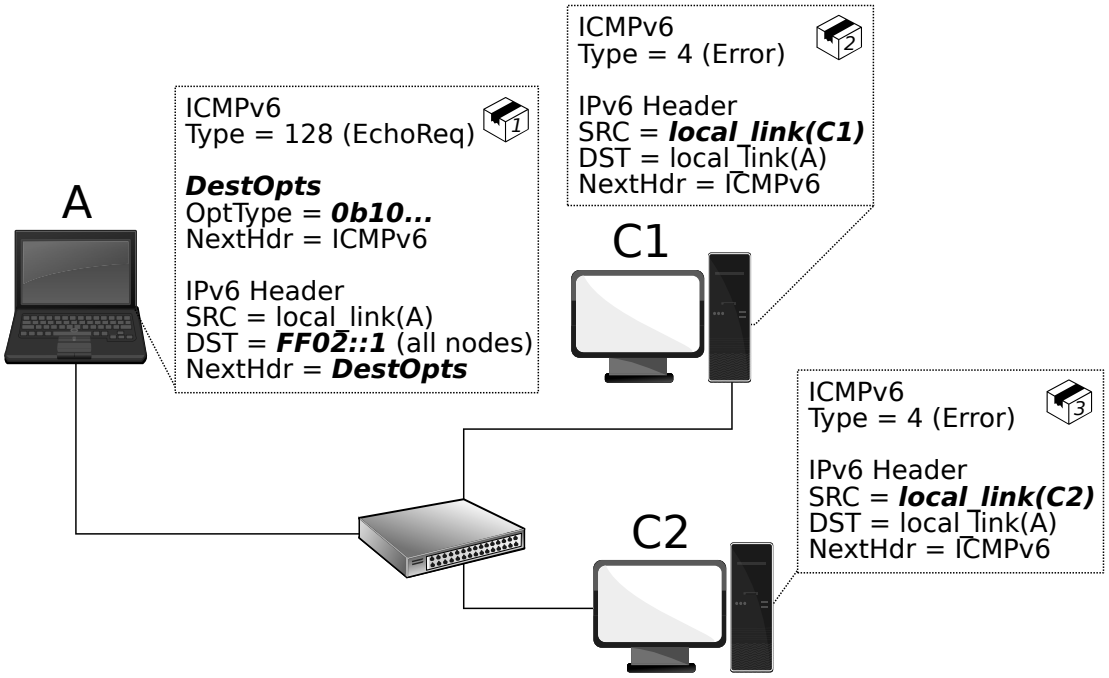

Figure 2.5: IPv6 Multicast Alive Scan

its own (source link layer) address (*slla*) and a randomly selected local network prefix (unique local address – *ULA*) which prompts the client to start auto-configuration (*A*=1) with the option to obtain other parameters (*O=1*) via DHCP. (4) The receiving client selects an address that contains the ULA prefix and its own MAC address or a random 64-bit postfix. The client validates the uniqueness of the selected address via IPv6 neighbor solicitation and (5) sends a *DHCPv6 solicit* request to obtain other parameters (in particular DNS servers) [33]. (6) In response, the attacker provides its own IPv6 address via DHCPv6 as the DNS server. The DHCPv6 part is similar to the DHCPv4 attack, but in the following step, the variants differ in detail. (7–8) Because the attacker is typically the only IPv6 router on the network, he/she can respond to DNS queries with any IPv6 address. The subsequent IPv6 traffic is then routed through the attacking computer. A variant of this attack[3] uses, for example, network address translation (NAT) and protocol translation [102, 103] to translate IPv6 into IPv4 packets which are routed to their regular target with the possibility to manipulate the intercepted request/response.

Firewall Circumvention with IPv6 Fragment Headers. The original purpose of fragmentation has been to forward packets whose total length is larger than the maximum transfer unit (MTU) of a network. The main problem with defragmentation lies

[3]https://wirewatcher.wordpress.com/2011/04/04/the-slaac-attack-using-ipv6-as-a-weapon-against-ipv4/

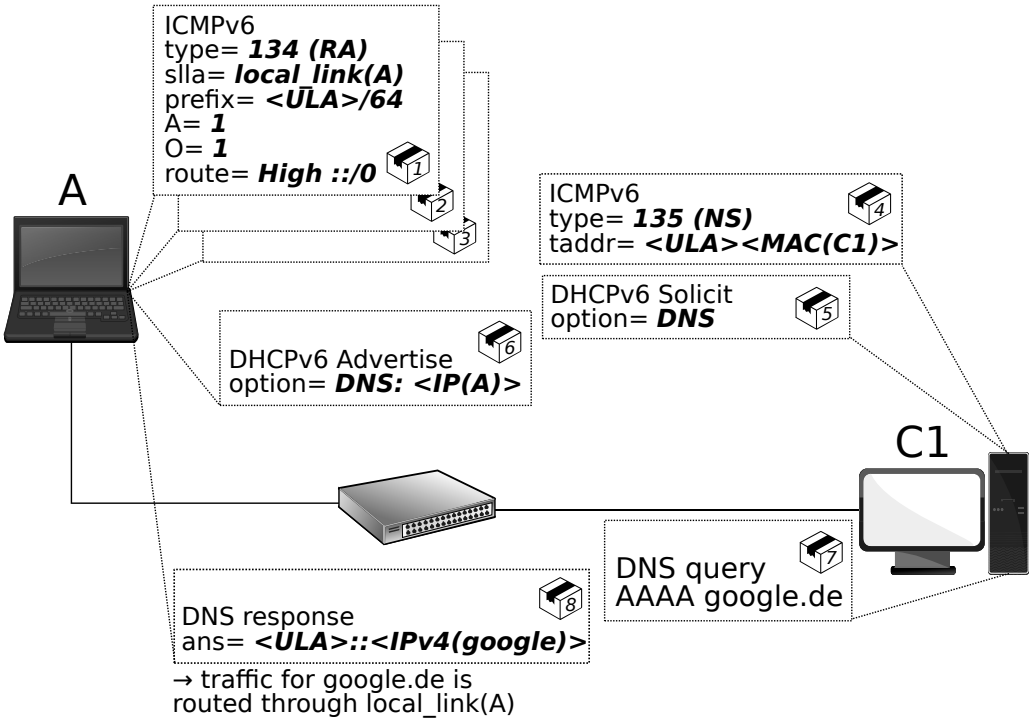

Figure 2.6: Router Advertisement, DHCPv6, and DNS Spoofing

in the fact that fragmentation offsets are only loosely checked for overlaps with the absolute limits (beginning and end) of the receiving buffer in the end systems. Overlaps *within* the buffer are not checked. Thus, overlapping offsets within a sequence of several fragments allow one to overwrite previous fragments. It is unpredictable whether the first or the second packet will be processed after receiving a duplicate at the receiver side if the two packets refer to the same (overlapping) offset. This opens up different vulnerabilities that are explained with the help of an example.

A potential attack that exploits the possibilities of IPv6 fragmentation is the hiding of the transmitted communication protocol. Figure 2.7 shows the IPv6 header extension principle used in this attack. Unlike IPv4, fragmentation is implemented in IPv6 as an optional extension header. This extension is reached via a list of references starting in the protocol header. In Figure 2.7 *Next Header 1* refers to the fragmentation header. The fragment offset in this header refers to the position of the fragment in the data packet. An offset of zero in the first fragment refers to the first extension header *after* the fragmentation header. Thus, protocol spoofing is possible. If the *Next Header 2* refers to a harmless extension (e.g., Destination Options (60)), then other headers, e.g. the transport protocol shift into the *Next Header 3*. An attacker can construct two fragments with the following assignments: the first fragment with the *Fragment Offset* = 0, *Identification* = 0, and *Next Header 3* = 58 (ICMPv6) and the second fragment containing the *Fragment Offset* = 0, *Identification* = 0 and *Next Header 3*

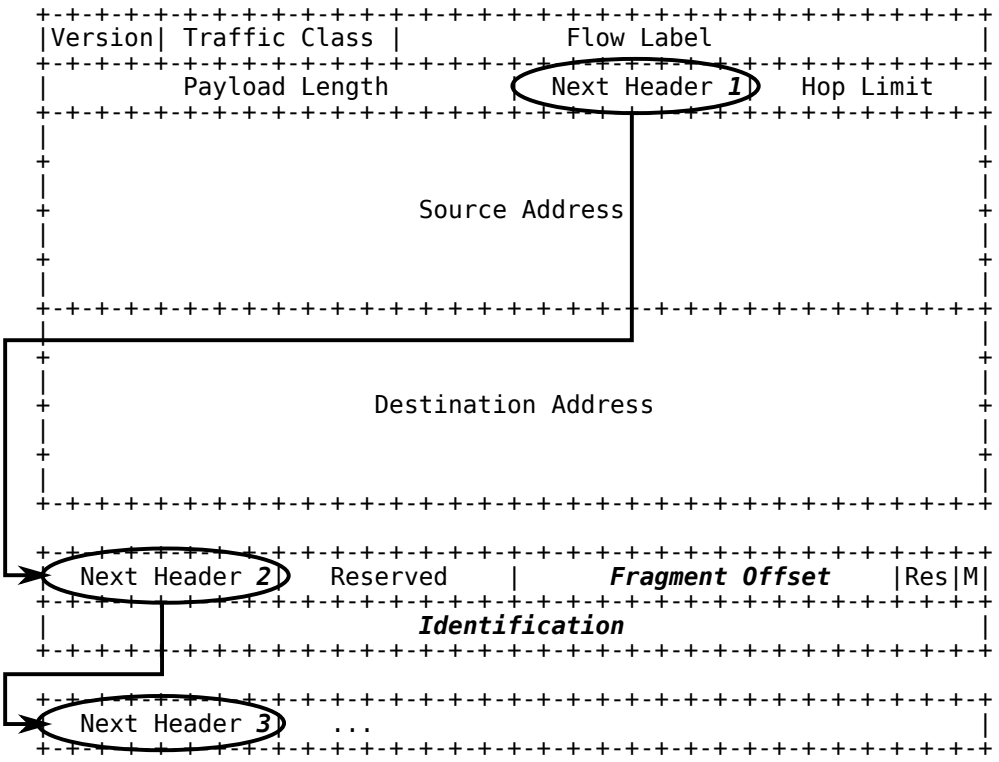

Figure 2.7: IPv6: fragmentation as extension header

= 6 (TCP). For pure packet-based processing, the firewall can only access the *Next Header 1 and 2* fields and at best check the *Next Header 3* field of the first fragment. If the firewall policy permits ICMPv6 Echo requests, then the second fragment, which overrides the first fragment, could pass unnoticed.

2.3 Approaches to Ward Off LAN Attacks

There are various approaches to protect physical and virtual networks, at least partially, from these attacks. The following subsection presents some of these approaches for physical networks and also has a look on solutions for virtual systems.

Approaches for Physical Networks

In order to secure classical physical systems, the networks are often divided into smaller segments. This can be done on a logical level by configuring IP subnets, but this is only a very weak division, or physically by an appropriate Ethernet wiring of the systems. At the transition points, e.g., routers or switches, data can then be analyzed

by packet filters. Packet filters [104] are the fundament of traditional network- and host-based security measures, but they are cumbersome to manage in large networks – a weakness that distributed firewalls [105] do not have (centralized policy design, distributed enforcement of policy). Distributed firewalls demand, however, implementations for each operating system and are themselves an attractive attack target due to their presence on the host.

Virtual LANs (VLANs) represent another possible security measure. They can be used for separating clients, so that only the systems that are associated with the same VLAN can communicate with each other. However, there are also attacks that aim at the so-called VLAN hopping [106]. Moreover, the use of VLANs is quite inflexible, since a client is either assigned to a particular VLAN or to none at all.

Another security measure is the use of intelligent switches that provide some protection against rogue DHCP servers and ARP attacks. Smarter L3 switches use, for example, DHCP snooping against DHCP attacks to enforce a fixed mapping between IP, MAC, and switch addresses/ports [106]. ARP Spoofing is prevented in this context by the discarding of non-approved source addresses. On some switches, this security measure can be circumvented by another attack which uses the spanning tree protocol (STP) to redirect traffic to an attacker. Such attacks can be avoided by limiting STP to ports which are explicitly used for switch coupling. Currently no security measures are known that reliably protect against similar IPv6-based attacks. In addition, there are problems with port-stealing attacks that override the internal cache of the switches with fake MAC/port pairs.

With the progressive development of software-defined networking (SDN) that centralizes the network control logic, projects like SANE [107] and Ethane [96] emerged. All of the complex functions – routing, naming, firewall policy specification, and checking – are managed and performed by a central controller. The main focus of these projects has been on access control and enforcement of communication relations rather than preventing L2/L3 attacks. In addition to these approaches for attack prevention, there is also some work for anomaly detection using SDN capabilities. Mehedi et.al. suggest several anomaly detection algorithms [108] that have been implemented on the basis of the NOX controller[4], and Ying Zhang published in [109] adaptive flow counting methods to detect additional anomalies in SDN-based networks. FRESCO [110] and OrchSec [111] provide beyond anomaly detection additional signature-based analysis methods, that can detect ARP cache poisoning, DDoS attacks, and DNS amplification attacks. Due to the central management by the controller, port-stealing attacks cannot be run within a software-defined network. IPv6-based attacks, however, are neither recognized nor prevented by the existing methods.

[4]https://github.com/noxrepo/nox

Approaches for Virtual Systems

Many of the above approaches cannot be readily transferred to virtual systems (VM hosts) because virtual machines cannot physically be separated from each other. The use of intelligent hardware switches is also not possible. Therefore, special, mostly proprietary software solutions are offered by the manufacturers of virtualization solutions. In [105] the concept of distributed firewalls was presented. It consists of modules of virtual firewalls that are located on the protected systems and which can be centrally administered. VMwall [112] is another application-level firewall for the Xen hypervisor that correlates VM traffic with process information using virtual machine introspection. Thus, it is able to block traffic based on the process that is sending or receiving the given packet. The simplified administration of both approaches is paid with an increased effort for the initial set up because the firewall modules must be installed on each system. Moreover, only those systems can be protected, for which there is an appropriate firewall module.

2.4 An Approach to Protect Switched LANs and Virtual Machine Networks

In order to prevent attacks on switched LANs this section proposes an approach to separate the host systems from each other. The firewall functionality for the hosts should be implemented within the network itself, i.e., on the intermediate system – the switch – and inside of a firewall module to which security-critical packets are redirected. The proposed approach applies the software-defined networking (SDN) paradigm. SDN provides the ability to separate the control and the data plane in a switch. As a result, the switch logic can be outsourced to a separate controller. Decisions regarding packet forwarding will no longer be made autonomously in the switch, but are passed to the central controller. The approach requires no changes to the host systems, and effectively prevents attacks on the layers 2 and 3. It can be easily and efficiently implemented using the OpenFlow protocol[5] [69] that will be introduced in the next subsection.

Introduction to OpenFlow

OpenFlow is a capture protocol that is widely used between the control and data plane of SDN-enabled switches. Figure 2.8 illustrates the structure of an OpenFlow switch and the operation of the OpenFlow protocol. The minimum configuration of a switch includes a flow table that defines the actions to be executed on data

[5]for current specification, see: https://www.opennetworking.org

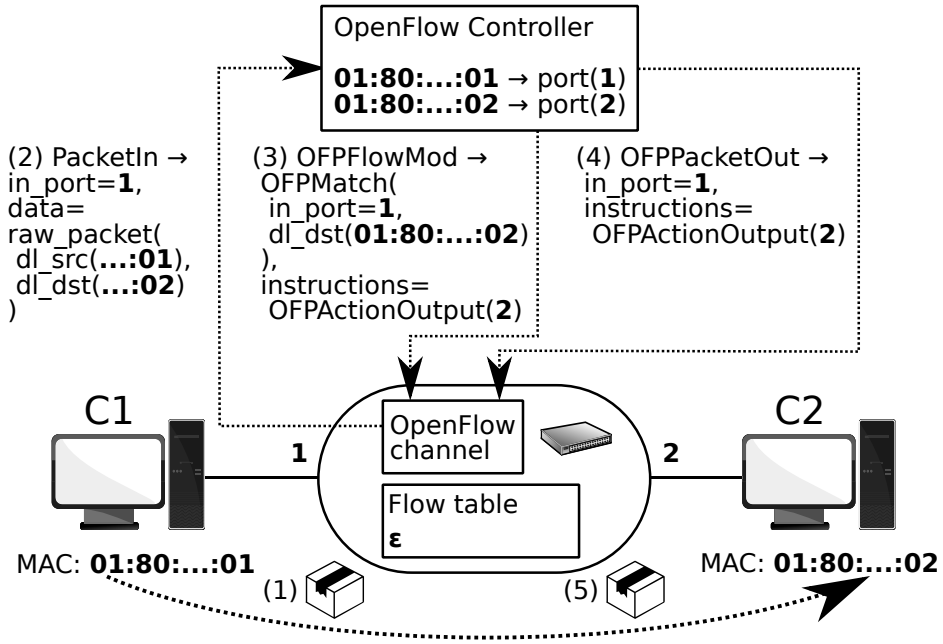

Figure 2.8: OpenFlow overview

flows and a secure channel to an OpenFlow controller that is responsible for making decisions about individual data flows in the network, i.e., to manage the flow table. The OpenFlow protocol is typically handled through a TLS encrypted channel. The flow table internally contains several fields that reflect the different frame and packet fields, such as source and destination hardware (MAC) address, layer-3-protocol, source and destination IP, and source and destination ports of UDP/TCP. Initially, this table is empty. At the beginning the switch establishes a connection to the controller that is typically not covered by the OpenFlow protocol procedure.

A typical OpenFlow protocol sequence is as follows. (1) When a packet arrives at the switch, it checks whether the fields in the packet matches a flow table row. (2) If there is a mismatch or the table is empty, as shown in Figure 2.8, a default action is executed. Typically, a *PacketIn* message is sent to the controller. The message includes the switch port, on which the packet is queued, a queue/buffer ID (not shown in the figure), and a copy of the packet (or a part thereof). (3) The controller determines what to do with the packet. In case of a simple MAC-learning switch, the port for the appropriate target system connected to the switch is found. Then, a rule is installed by means of an *OFPFlowMod* message in the switch that forwards all further packets of the same data flow within the switch from the source to the target system. The message consists among others of an *OFPMatch* data structure which contains all the fields that need to be matched in subsequent packets and a set of instructions that specify what to do with the matched packets. For the illustrated (MAC-learning) example, *OFPActionOutput* instructs the switch to output the packet to the port of

the target system. (4–5) Finally the controller has to decide what to do with the packet in the queue that has triggered the original PacketIn message. The *OFPPacketOut* message instructs the switch to forward it to the target system.

OpenFlow-based Secure Switching and Firewalling

To provide security functionality in switched LANs the data streams (or a part thereof) have to be forwarded to a security module. This approach can easily and efficiently be implemented using software-defined networking. The above-mentioned widely used OpenFlow protocol can be used as a capture protocol. First considerations for this area were presented in [113].

Figure 2.9 shows the principle of the approach. It uses an OpenFlow-enabled switch as the data forwarding component. The data plane switch is connected via the OpenFlow protocol with the control plane, in which various security-critical services are implemented. By shifting the switch logic into this separate controller, packet forwarding decisions are made in a policy-based software switch inside this controller and not in the data plane switch. Thus, the software switch gains complete control over the network and the data routing. This concept also allows that certain security functions, which are usually applied in firewalls, application-level gateways, and complex security configurations, can now directly be taken over by the software switch. Thus, hosts can optionally be authenticated using a port authentication service. In addition, the proposed approach installs rules in the OpenFlow switches ensuring that all address configuration and resolution packets are redirected to the controller and not distributed further. In order to still be able to fulfill the purpose of the respective protocols a corresponding address configuration and address resolution service has to be integrated into the controller that checks the meaningfulness of ARP requests and sends the corresponding ARP replies. As a result, ARP poisoning attacks can be prevented. In contrast to previous work, this concept considers also the IPv6 address configuration and address resolution protocols to prevent similar spoofing attempts using the next generation of IP. An even higher degree of security can be achieved by the separation of network segments by means of physical cabling, configuration

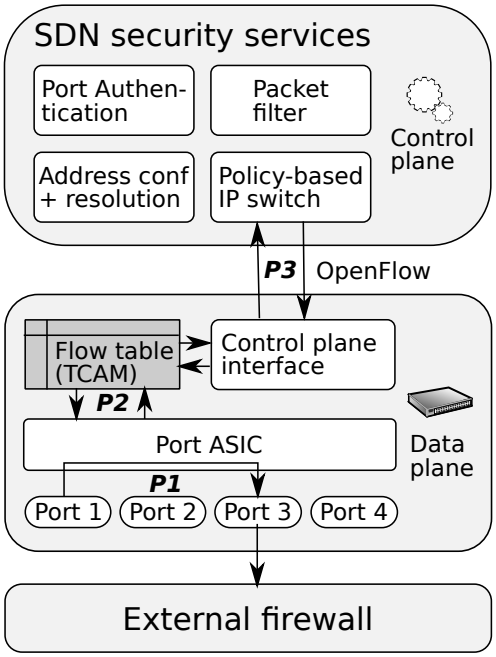

Figure 2.9: Approach for SDN-based security services

of subnets, and routing. At least the subnet-based network separation and a secure routing can be implemented also using the SDN paradigm. The policy-based switch enforces a separate subnet for each client. The resulting network is based on *switched routing* which uses auxiliary information from the address configuration services to enforce a strong binding between a packet and its origin as well as its target. The corresponding concept is referred to as *IP switching*. Section 2.5 describes these security functions/services in more detail.

In addition to the depicted security services, further security functionality, e.g., deep packet inspection, can be integrated into the controller. Here, however, the following problem occurs. The OpenFlow data plane works internally flow- and not packet-oriented, i.e., each incoming packet for which there is no flow rule in the flow tables is redirected to the controller (see slow path *P3* in Figure 2.9) which causes a high overhead. Only after the controller has stored a corresponding flow rule in the flow tables of the affected OpenFlow-enabled switch, subsequent packets can be forwarded to the respective ports (see fast paths *P2* and *P1* in Figure 2.9). For a packet filter, the high overhead for the first packet of a flow is not critical. A deep packet inspection, in contrast, must examine each packet or a sequence of consecutive packets of a flow. To avoid the expensive sending of packets to the controller this approach stores a specific rule in the flow table of the data plane switch that directs security-critical packets for DPI to a separate external firewall.

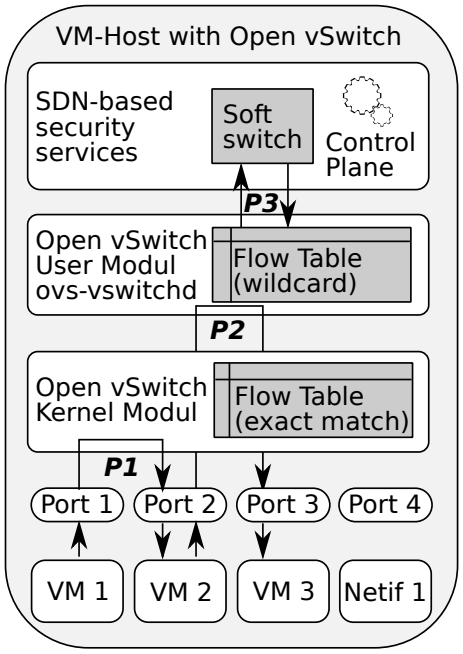

Figure 2.10: SDN-based virtual machine switch

Secure Data Exchange between Virtual Machines. As a result of the increasing virtualization of computer systems, areas arise in corporate networks and cloud environments that represent blind spots for the network monitoring. Conventional firewall systems cannot protect virtual machines (VM) because the communication between the virtual machines runs only within the virtualization server/host. Therefore, virtualized systems, in contrast to physical systems, are particularly susceptible to the above introduced attacks on layers 2 and 3. Firewalls within a virtual machine do not adequately solve this problem, since the compromise of the virtual machine at the same time implies the compromise of the firewall contained therein. Thus, the entire host along with all guest systems is at risk. Furthermore, they significantly affect the performance of the virtual environment and require additional manual configurations in each

virtual machine. The hypervisors of the virtualization servers, however, have often similar characteristics and identical configurations, which would allow an integration of firewalls outside of the virtual machines. For this purpose, sensors are necessary, which record the network traffic between the virtual machines and send it to an outside firewall. This can also be achieved by means of the SDN paradigms by using an OpenFlow-enabled switch – the Open vSwitch [114, 115] – as a redirector component. The Open vSwitch provides a virtual data plane for virtual machine hosts and supports all major versions of the OpenFlow protocol.

Figure 2.10 shows the approach for virtual machines. Open vSwitch comprises two components – a userspace daemon (ovs-vswitchd) and a datapath kernel module. The userspace daemon is connected with the control plane. It implements the same security services for the physical switch as above. The control flow path *P3* is the same as for the physical concept (cf. Fig. 2.9 and Fig. 2.10) and also uses the OpenFlow protocol. The control flow and data path *P2* resembles the OpenFlow approach insofar, as the ovs-vswitchd manages the kernel module in a similar way like the control plane manages the userspace daemon. The flow table in the userspace daemon can compare all frames, packets, and segment fields defined in the OpenFlow standard – a degree of flexibility which goes hand in hand with a loss of speed. Therefore, there is a further datapath *P1* in the kernel module which compares the frame/packet header bit exactly against a microflow cache and directly forwards the packet in the case of a hit to the cached target. In this way, the path *P1* resembles the appropriate path through the application-specific integrated circuit of a physical switch (port ASIC, cf. Fig 2.9) that is also limited to a bitwise comparison of specific fields from layers 1–3. If a comparison is required that masks sub-fields a detour of the packet on the path *P2* is needed that allows for wildcard comparisons across all fields. This path is thus similar to the corresponding path *P2* of physical switches via a ternary content addressable memory module (TCAM, cf. Fig. 2.9). A TCAM is an associative memory that compares input data against its content and returns the corresponding address of a hit within a guaranteed time interval. The word ternary means that within the store in addition to the two states of 0 and 1, a third "Don' t care" condition matches with every input bit. This is used, for instance, in switched routers for comparing IP addresses against network masks in which not all bits of the IP address are necessary for determining the target port.

2.5 SDN-based Security Services

The attacks described in Section 2.1 result essentially from the decentralized management of critical network services. These attacks can be prevented by means of centralized SDN-based services for security-critical operations.

Authentication at Switch Ports

An important problem for the implementation of the above introduced approach is the authentication of the hosts and/or virtual machines for accessing the switch and thus the network. A good selection criterion for an authentication service based on one of the "classical" services, e.g., TACACS [116], TACACS+[6], EAP [117], and RADIUS [118]), is the ease of its integration into existing network environments. The use of already existing and tested components, such as authentication servers or directory services, increases the security and flexibility of a solution. Therefore, this approach applies the extensible authentication protocol (EAP) [117], which is often applied in wireless networks and point-to-point communications. The existing authentication infrastructure of most corporate networks in the form of RADIUS servers and LDAP [119] directory services can directly be used for a compatible SDN-based authentication service based on EAP.

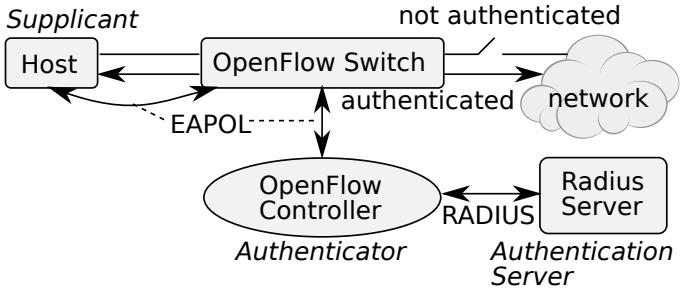

Figure 2.11: EAP/RADIUS authentication using SDN/OpenFlow

The EAP standard is a port-based access restriction for switch ports that only unlocks after a successful authentication. The switch acts as an intermediary between the host (supplicant) and the authentication server – usually a RADIUS server – that optionally queries a directory service (see Figure 2.11). The problem to be solved is the integration of two different authentication formats in a SDN-based network. Although all parties, i.e., the hosts, the switch, and the RADIUS server, communicate with each other via EAP, the EAP packets are encapsulated differently, e.g., in Ethernet frames (EAP over LAN (EAPoL)) or UDP packets (RADIUS over UDP). The solution is that only the EAPoL packets are allowed with a non-authenticated switch port (ether type = 0x888E, address = 01:80:C2:00:00:03 multicast). These are identified by the OpenFlow switch and redirected to the OpenFlow controller. The OpenFlow controller sends the EAP messages as RADIUS packets to an authentication server. If authentication succeeds the controller transfers appropriate rules to the OpenFlow Switch allowing the host to participate in the communication. *Hostapd*[7] is used as authentication server. It is a daemon running in user mode with software-Wi-Fi access point functionality. In addition, it provides an EAP authenticator as well as a RADIUS client and server. The EAP standard supports various authentication protocols. For a prototype of this approach, the EAP-MD5 [117] protocol was selected.

[6] http://tools.ietf.org/html/draft-grant-tacacs-02
[7] http://w1.fi/hostapd/

In practical use, more secure alternatives, such as EAP-TLS, EAP-TTLS, or EAP-PEAP, should be deployed. In the SDN-controlled network, however, the broadcast traffic is blocked and the communication partners communicate in a circuit-switched manner. Clients are unable to analyze or modify packets that are not addressed to them (even in promiscuous mode). Thus, EAP vulnerabilities that primarily relate to networks that use a shared medium, such as the IEEE 802.11 wireless networks or the classic Ethernet, cannot be exploited.

Spoofing and Scan Resistance with Address Configuration and Address Resolution Services

Many of the LAN security issues result from the fact that two different layers are responsible for addressing a system: layer 2 and 3, which have to be reconciled. To obtain the corresponding MAC address of a system with an IP(v4/v6) address, ARP and ICMPv6 Neighbor Discovery (ND) are used. One of the major problems in local area networks is that address configuration and address resolution frames are broadcasted within the network. These frames can be monitored from any network participant and responded. This enables an attacker to bring the system into an inconsistent state, to eavesdrop on running network connections, or to suppress further communication. The following address configuration and address resolution schemes implement the necessary functionality to prevent various aspects of these attacks.

IPv4 address configuration service. Spoofing can mainly be avoided by means of address configuration services. The IPv4 address configuration service of the proposed approach is based on a controller-internal DHCPv4 service that redirects all DHCP client packets according to a rule in the switches to the controller (see rule DHCPv4c in Table 2.1). The contents of the rule table can be interpreted as shown in Figure 2.12. The header of the table corresponds to the match and action data structures defined in the OpenFlow standard that are transferred initially to the controller (lower left part of the figure). The prefixes *OFPXMT_OFB* and *OFP_ACTION* have been omitted from the table and the suffixes *eth_type*, *ip_proto*, *udp_src*, and *udp_dst* correspond to the grey marked fields of the protocol stack shown in the upper part of the figure. The switch-internal representation of the match data structures is not defined in the OpenFlow standard. In physical switches it is typically converted into a sort of bit mask for a TCAM (lower middle part of the figure). If a packet matches this mask it is sent to the port defined in the OUTPUT_ACTION (controller in the lower right part of the figure). The internal DHCPv4 service of the controller assigns IP addresses or renews leases for the connected systems. Allocated address entries are kept in the configuration of the controller for the address resolution services which are discussed below. This measure prevents attacks from rogue DHCPv4 servers because a spoofing is not possible without knowledge of the 32 bit DHCP transaction ID and the exact

time of the client requests. An attacker does not even see the DHCP requests of the other systems required to send the malignant answer in the right moment.

RULE	eth_type	ip_proto	udp_src	udp_dst	output_port
DHCPv4c	0x0800 (IPv4)	17 (UDP)	68 (client)	67 (server)	controller
ARP	0x0806 (ARP)				controller

Table 2.1: IPv4 address resolution service rules

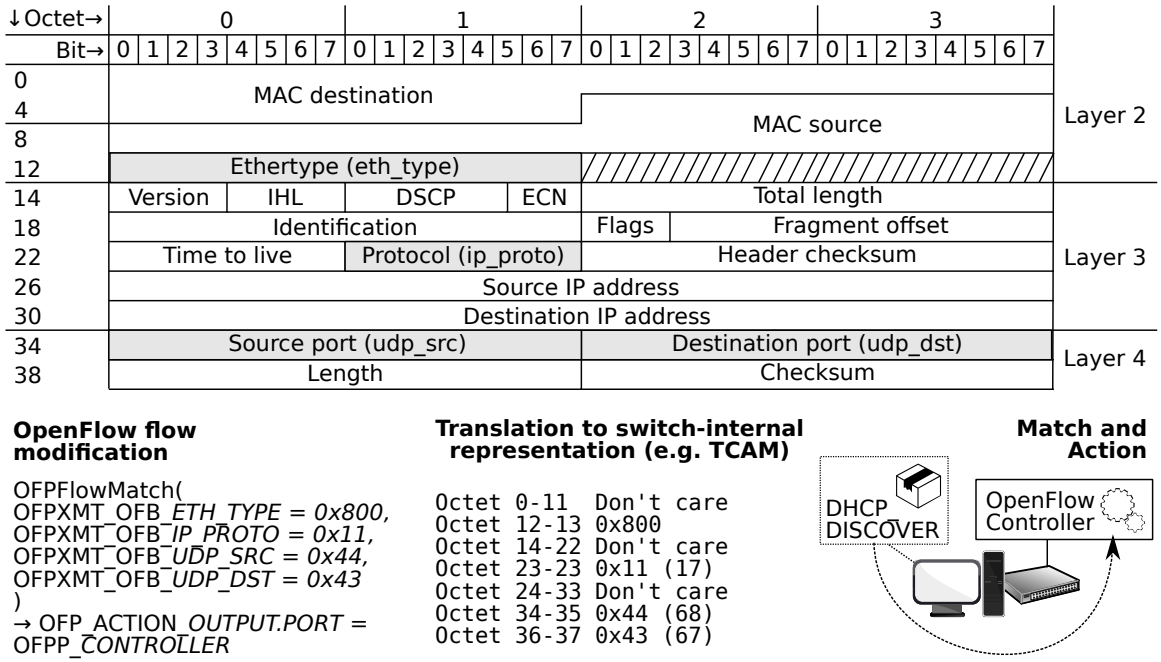

Figure 2.12: Protocol switching

IPv4 address resolution service. Countermeasures against network scans and some more anti-spoofing measures can be implemented in the address resolution services. The IPv4 address resolution service is based on the knowledge of the DHCPv4 address configuration service. The ARP service part installs rules in all switches that redirect Ethernet packets with the ethertype 0x0806 (ARP) to the controller (see rule ARP in table 2.1). In this way, no ARP packets are forwarded within the OpenFlow-enabled switches. ARP responses are transmitted exclusively from the OpenFlow controller based on the knowledge of the DHCPv4 service. Additionally, ARP requests that are not related to the gateway address in the appropriate subnet of the requesting client are ignored silently. These measures effectively prevent attacks, such as ARP scans, ARP spoofing, and ARP flooding.

IPv6 address configuration services. IPv6 address configuration is provided by an internal ICMPv6 router advertisement (RA) service and a DHCPv6 service linked to the DHCPv4 server. The ICMPv6 router advertisement service enforces configuration of IPv6 addresses using DHCPv6. This is done through regular flooding on all switch ports using router advertisements with the managed address configuration flag set. The DHCPv6 service works initially similar to the DHCPv4 service by redirecting DHCPv6 messages to the controller (see rule DHCPv6c in Table 2.2). DHCPv6 requests are answered exclusively using the IPv4-mapped IPv6 address of the requesting client [120] based on the knowledge from the DHCPv4 service. The IPv4 address 192.168.120.63, for example, is mapped into the IPv6 address ::FFFF:192.168.120.63. DHCPv6 spoofing is therefore even more limited than for DHCPv4 because in addition to the lack of knowledge about transaction IDs or the time of client requests there is also no possibility to exhaust the address pool. Another advantage of this mapping is that the communication can be processed by existing firewall logic, which is derived from the IPv4 firewall rules. The disadvantage of non-routable IP addresses associated with the mapping may be offset by a very simple form of NAT which applies a direct (IPv6-)address to (IPv6)-address translation (static one-to-one address assignment [36] without protocol/port translation) at the edge of the network.

RULE	eth_type	ip_proto	udp_src	udp_dst	output_port
DHCPv6c	0x8DD (IPv6)	17 (UDP)	546 (client)	547 (server)	controller

Table 2.2: IPv6 address configuration rule

IPv6 address resolution service. The ICMPv6 neighbor discovery service is based on the same idea as ARP for IPv4. Two flow rules are installed on all switches which redirect ICMPv6 neighbor discovery solicitations and neighbor discovery advertisements to the controller (see rules ICMPv6nds and ICMPv6nda in Table 2.3). All solicitation requests except those to the IPv4-mapped address of the gateway in the subnet of the requesting client are ignored silently. This method prevents standard IPv6 scans as discussed in Section 2.2. Additionally, there could be another hole which allows scans in the form of multicast listener discovery [121, 99] messages. The analysis of these messages, however, is still a subject of current research. The best option seems to be to completely disable multicast listener discovery by installing corresponding rules.

RULE	eth_type	ip_proto	icmpv6_type	output_port
ICMPv6nds	0x8DD (IPv6)	58 (ICMPv6)	135 (solicit)	controller
ICMPv6nda	0x8DD (IPv6)	58 (ICMPv6)	136 (advertise)	controller

Table 2.3: IPv6 address resolution rules

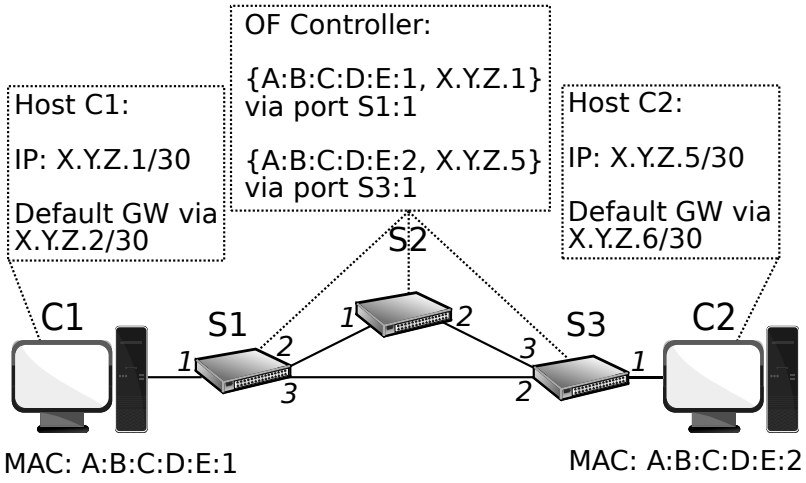

Figure 2.13: IP switch configuration with virtual gateway addresses

IP Switching with Topology Hiding and Routing Enforcement

The aforementioned security services provide a *secure initialization of the network configuration* but *no secure message routing*. They prevent that data connections are kidnapped by means of ARP spoofing or corresponding IPv6 attacks, but packets can still be hijacked if a false identity (MAC address) is adopted by a host and only a simple MAC-learning switch logic is applied in the OpenFlow controller. In addition, broadcasts like in ARP are a major problem as already mentioned in the Ethane approach [68] because they flood the controller too much. Moreover, broadcasts and ARP services can be used to explore the network topology at the link. Therefore, a service is necessary, which (1) binds the identity of a system to a switch port, (2) limits the number of possible requests for other systems, and (3) realizes the route of the data connection between two systems in a secure way.

Historically broadcasts have always been limited by separation of networks. The binding of a system to a switch port is also a special case of network separation in some way. Therefore, the objectives (1) and (2) can be achieved by network separation by means of address configuration. For this purpose, the address configuration service assigns a private /30 subnet to each host using DHCPv4. This addressing scheme allows one to hold four IP addresses for each subnet. The lowest address is the network address. The next two addresses are assigned to the host and a virtual gateway that references the OpenFlow controller. The upper address remains for further use, e.g., as the IPv4 broadcast address [122]. The hosts in the subnets are forced by this addressing scheme to use routes via the virtual gateway instead of direct communication to reach the target systems. The only possible broadcast request – an ARP request for the virtual gateway address – is intercepted by the OpenFlow controller and not flooded.

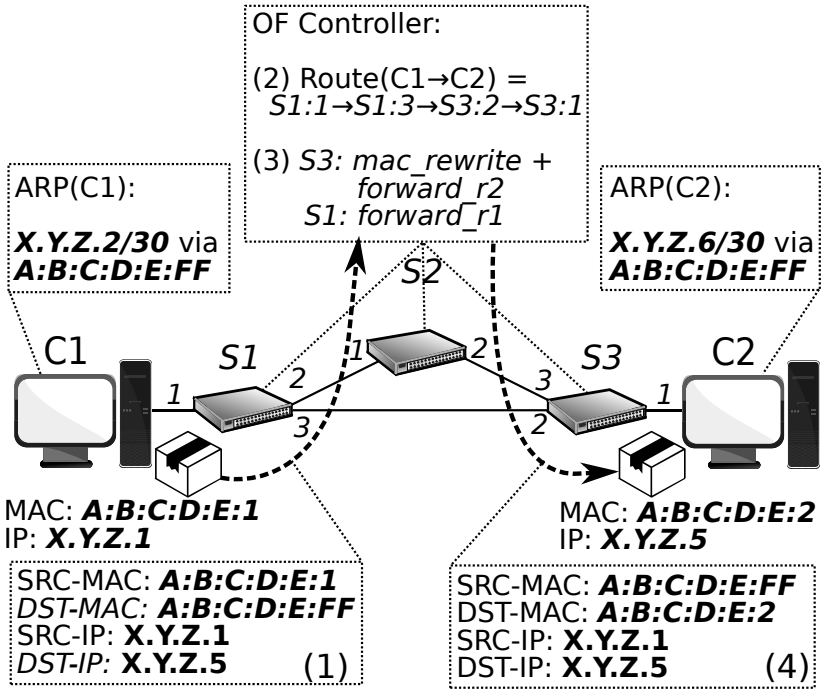

Figure 2.14: IP switching (a.k.a. routing) with virtual MAC addresses

Figure 2.13 exemplifies this address configuration. The address resolution service in the controller assigns the host *C1* to the network *X.Y.Z.0/30*. The resulting address configuration (IP(C1) = X.Y.Z.1, GW(C1) = X.Y.Z.2) are deposited together with the MAC address of the host in a database of the controller. The next available subnet (X.Y.Z.4/30) is assigned to host *C2* and its configuration is also stored in the database. The hosts know only their own IP address and the respective virtual gateway, thus limiting their ability to scan for other hosts with layer-2 protocols. Only the IP addresses of the other systems are known, and the actual network topology remains hidden.

Based on the information of the (subnet) address configuration service, a secure routing service is established, which binds the identity of the participating systems for the duration of a connection to their source and destination switch ports to prevent kidnapping of data. The routing process is exemplified in Figure 2.14 (based on the address configuration from Figure 2.13). The virtual gateway address is the only address that is resolved via ARP or ICMPv6-ND (see MAC address *A:B:C:D:E:FF* in the ARP caches *ARP(C1)* and *ARP(C2)* of the figure). (1) When sending IP packets, the hosts use this address as the destination MAC address (*DST-MAC*) together with the IP address (*DST-IP*) of the target computer. The OpenFlow-enabled switches forward the first packet of a connection to the controller. (2) The controller determines a suitable route based on the source and target IP addresses (switch 1, port 1 (*S1:1*) via switch 1, port 3 (*S1:3*), via switch 3, port 2 (*S3:2*) to switch 3, port 1 (*S3:1*)).

(3) Appropriate match entries are created in the switches (see Table 2.4, *nw_src* = SRC-IP, *nw_dst* = DST-IP). The rules are installed in the reverse transport direction for the packet (first *mac_rewrite+forward_r2* on switch S3, then *forward_r1* on switch S1) to avoid multiple redirects of the same packet by the subsequent switches to the controller. The last switch on the route is given the task to rewrite the MAC addresses (*set_field* part of the *mac_rewrite+forward_r2* rule) so that they are valid for the target system. (4) The queued packet is released (by forwarding to switch 1, port 3 – S1:3) and reaches the target in accordance with the defined rules. A glance at the chain created in this way in Table 2.4 illustrates the binding of each packet to its source and destination. In switch *S1* the source IP address *X.Y.Z.1* is bound by the rule *forward_r1* for the exemplary data flow to input port (*in_port*) 1. The rule *mac_rewrite+forward_r2* for switch *S2* rewrites the virtual destination MAC address to the MAC address of host *C2* and binds the data flow and the destination IP address to *output port 1.* Thus, a spoofing of IP addresses is prevented.

RULE	in_port	eth_type	nw_src	nw_dst	set_field	output
forward_r1	1	0x800 (IP)	X.Y.Z.1	X.Y.Z.5		port 3
mac_rewrite +forward_r2	2	0x800 (IP)	X.Y.Z.1	X.Y.Z.5	eth_dst = A:B:C:D:E:2	port 1

Table 2.4: Routing rules for Figure 2.14

In-Network Firewalling, Robustness against Firewall Bypassing, and Connection of External Firewalls

Physical switches often implement simple firewall logic in the form of access control lists – a concept that is also possible with the SDN paradigm. Firewalls can be implemented as a combination of SDN-based rules on the switches and controller-based firewall logic. Simple packet filtering can directly be implemented by corresponding rules on OpenFlow-enabled switches. In addition, firewalls with stateful connection management can be implemented on the switches with some help of the controller and the dynamic creation of rules. IP switching is basically a flow-based switching with packet filtering functionality. For each new connection, a request is sent to the controller which then decides whether this connection should be permitted or not. After this a specific rule is stored in the switches ensuring that the other packets of this connection no longer need to be treated by the controller. All these security rules can individually be deposited in the controller for each client and are enforced independently of the switch port to which the client is connected to. Each port on each SDN switch thus becomes a firewall.

Important security measures that should be contained in every firewall approach are static firewall rules which are directed against the circumvention of the firewall logic. The concept described above, for example, allows a filtering on packet basis, but

RULE	eth_type	ipv6_exthdr	ip_proto	icmpv6_type	output
fragment	0x8DD	44			controller
hop_by_hop	0x8DD	0			controller
routing	0x8DD	43			controller
destination	0x8DD	60			controller
mobility	0x8DD	135			controller
icmpv6_redirect	0x8DD		58	137	controller

Table 2.5: Static rules against firewall bypass

fundamental problems that affect pure packet-based firewall logic, e.g., the processing of fragmented oversized packets, as supported by IPv4 and IPv6, cannot be solved. Past experience has shown that it is not possible to implement packet fragmentation in a secure (and at the same time efficient) manner. For this reason, the presented approach installs rules on all switches, which discard IPv4 and IPv6 packets with fragmentation options/headers (see rule *fragment* in Table 2.5). In addition, IPv6 at this stage represents a good circumvention method of local security policies. Another focus of defensive measures against controller bypass are, therefore, the problematic IPv6 extension headers. The following headers should be redirected through rules on the switches to the controller (to log attack attempts) or silently discarded: *hop-by-hop options*, *routing header*, *destination options*, and *mobility header*[8] (see corresponding rules in Table 2.5). Although these are almost all of the IPv6 extension headers, these rules are in line with recent observations of transit networks[9] which discard packets with the same headers. They represent special use cases that are often not supported or configured, but are, however, used within the IPv6 attack suites as an evasion method for IPv6 security measures. Another packet type that should be blocked is ICMPv6 redirects (see rule *icmpv6_redirect* in Table 2.5). This packet type allows another attack in which a client can be redirected to a fake gateway address.

If external firewalls are integrated in the software-defined network, static or dynamic rules in the switches allow one to route security-critical connections dynamically and transparently to the respective firewall, application-level gateway, or intrusion detection system. From there, only after successful filtering, the data packets are sent to the target system.

2.6 Evaluation

As part of the research for the described concept, an OpenFlow controller that implements the described functionality was implemented with the help of the Ryu frame-

[8]The mobility header supports Mobile IPv6 [123].

[9]`https://tools.ietf.org/html/draft-ietf-v6ops-ipv6-ehs-in-real-world-02`

work[10]. During the development phase, the controller was tested extensively with Mininet [124]. The proposed defensive measures against ARP scans, ARP spoofing, port-stealing, and DHCPv4 spoofing with DNS spoofing worked as expected. There were, however, mixed results for IPv6. Scans of the IPv6 test network, neighbor discovery spoofing, router advertisement spoofing, and DHCPv6 with DNS spoofing were successfully blocked. The defensive capabilities against malformed IPv6 packets, however, depend on the used IPv6 stack, e.g., the Linux IPv6 stack discussed below.

Evaluation of Controller Bypass Resistance against IPv6 Manipulations. To test the defenses against malformed IPv6 packets, two virtual clients in a test system were evaluated with the penetration test suite from the THC IPv6 toolkit[11]. One of the clients was acting as a test server to check which packets have passed through the firewall and the other client was used to send manipulated packets. The test suite consisted of 56 test cases that were first executed without the SDN-enabled switch to test the Linux-based IPv6 forwarding and after this with it including the controller prototype with all services of the presented approach. The test purpose was to prove whether the firewall can be bypassed (*pass*) or whether the packets are blocked (*fail*). 38 tests failed with the standard Linux stack and thus necessarily also with the Linux-based SDN-enabled switch. Among the tests that failed were also those with overlapping *fragment headers*. The structure of the test suite is interesting in itself. Approximately half of the circumvention tests started with a single extension header type and increased the number of types up to three. As expected, all standard test cases with *hop-by-hop*, *destination options*, and *source routing options* passed the linux stack (i.e., they were forwarded without complaints) and failed with the controller prototype (i.e., they were successfully blocked). However, there were test cases with multiple *destination option headers* in a packet. These passed the firewall policy of the controller prototype for unknown reasons. Three more test cases passed through the firewall because they were not covered by a policy and were also not efficient to implement: ICMPv6 echo requests (*ping6*) with bad checksum, zero checksum, and with hop count 0. All other test cases failed in the the controller prototype, in contrast to some variants with a *source route option* that passed through the default stack. A direct neighbor solicitation test was successful on the standard stack, but failed in the prototype because of the defenses against network scans.

2.7 Conclusions

This chapter has analyzed attacks on switched LANs and VM networks that are an integral part of internal and targeted attack phases. The IPv6 protocol that has always

[10] http://osrg.github.io/ryu/
[11] https://github.com/vanhauser-thc/thc-ipv6

been sidelined in previous research so far was also considered. A solution for SDN-based defensive measures against these attacks was introduced. The proposed SDN services include ARP and DHCP services which intercept the ARP and DHCP packets in the network and thus solve the broadcast problem of SDN, in which broadcast packets are redirected to the controller several times. The OpenFlow controller can then answer these requests as a central authority. A further service of the controller is the transparent secure routing of connections. Due to the flexibility of OpenFlow, the proposed approach can be used not only in physical environments, but also in virtual environments in which classical network security measures cannot be used so far. The OpenFlow rules can be bound to specific physical or virtual hosts/host groups and are applied independently of the switch port to which a host is connected. This is especially interesting for virtual environments, when a virtual machine migrates to a different host. Network scanning and spoofing attacks on layers 2 and 3 of physical and virtual intermediate or end systems have been found to be controlled well and prevented by the SDN switches and the developed security services. The proposed solution requires no changes to the host systems, but requires a SDN-based network infrastructure. This leads to some remaining research challenges explained next.

Cascading of in-band communication. There are two ways to connect a switch to a controller, in-band or out-of-band. Out-of-band communication realizes the switch-controller connection via a separate control network, which is not always desirable because of the wiring involved. In-band communication, however, is associated with the problem that the communication of cascaded switches to the controller is passed through the OpenFlow-managed network, which is not trivial to implement.

Mixed operation with classic switches. Since existing networks may be changed only gradually to SDN-based networks, mixed mode scenarios may appear with unknown security features. The question is which switches must be replaced first, and what security restrictions result from the remaining non-SDN switches? A problem in the mixed mode is, for example, the use of ARP to identify systems that are connected to the non-SDN switches. All systems connected to SDN switches can be identified by the address configuration service. The MAC addresses of the systems connected to a non-SDN switch must be resolved through ARP requests that may be forwarded under no circumstances to systems bound to an SDN-switch. In this case, the controller could be brought by forged answers into an inconsistent state which threatens the security of the overall system.

Rule interference. If more SDN applications are placed into the controller the rules of these applications may overlap with the ones installed by the security services. OpenFlow provides a priority system that can solve this problem, but unexpected interactions should be investigated in future work. One possible example of such an interference is a connection-based load-balancing algorithm which would compete with the shortest path algorithm for secure routing in this approach (see

Fig. 2.14). The proposed approach enforces a fixed path for each connection on all switches to prevent the kidnapping of data. A load balancing algorithm could, however, come to the conclusion that a connection via a longer route on less busy links is more optimal and install appropriate rules. Depending on the order of the installed sub-rules, these connections are either not protected by a secure routing against manipulation or the routing for a connection would not work at all.

Weakly- vs. strongly-typed Network OS. A part of the SDN functionality is achieved through so-called network operating systems (NOS), which provide frameworks for the programming of SDN applications and hide some of the complexity of the OpenFlow protocol and the network management. For this work, various NOS written in C/C++, Java, and Python were investigated. Python proved to be very efficient for a rapid prototyping. This efficiency was later paid with an increased effort in troubleshooting packets that disappeared mysteriously. A major problem with the troubleshooting was the weak type system of Python in which it is often unclear which objects are channeled through a function. Secure network operating systems should be based on strongly-typed languages like Java – which is used, e.g., in the Floodlight controller[12] – or if speed, readability and security are to be combined, for example, be realized in Rust[13].

On the other hand, software-defined networks offer opportunities to further improve the IT security. Some options for future work are discussed next.

Anomaly detection and reaction to attacks. The central overview of the network activities of different hosts in the SDN, as well as statistics that are collected directly on the switches, makes it possible to search for anomalies in the network traffic. For this purpose, obviously SDN-based anomaly detection techniques are necessary. If the detected anomalies indicate attacks (e.g., the outflow of large amounts of data through unknown protocols), a dynamic firewalling approach can be used which restricts the use of various network services by the (attacking) hosts. Visited web links by the attacking host could be diverted to an error page that informs the user that the computer is blocked, while mail connections to the administrator are still possible to discuss the issue. All other mail recipients and connections are blocked, too.

Dynamic firewalling. The above-discussed dynamic firewalling technique also allows the flexible use of pre-defined personal or role-specific policies in a SDN firewall application that dynamically allows/prohibits connections, while taking the client site (mobility assistance for multi-user systems as well as laptop and smartphone users) into account. In this case, the firewall rules are not tied to a specific device, but to the authentication of a specific user in the network.

[12] `http://www.projectfloodlight.org/floodlight/`
[13] `https://www.rust-lang.org/`

Wireless SDN. The network interfaces to wireless local area networks (WLAN) represent an additional blind spot for network monitoring. Often the Wi-Fi access points are not part of the monitoring and security infrastructure, but represent only the last hop to the (mobile) devices. This approach inevitably loses sensitive information of the communication between the wireless AP and the device. This causes that attacks remain potentially undetected. In the past, the wireless encryption was broken several times, e.g., for *wired equivalent privacy* (WEP)[14], *Wi-Fi protected access* (WPA)[15] and indirectly for WPA2 using the *Wi-Fi protected setup*[16] (WPS) protocol. Further attacks exploit the fact that management frames are transmitted unencrypted to de-authenticate individual clients[17] and to reconnect them using, e.g., rogue access points as man-in-the-middle. Integrating an intrusion detection concept directly into the WLAN APs is inflexible and technically difficult to implement. A software-defined networking approach which passes the Wi-Fi AP authentication and monitoring information to the SDN-based security services, in which an analysis of incoming data streams will be carried out anyway, seems more reasonable.

[14] http://www.dummies.com/WileyCDA/how-to/content/understanding-wep-weaknesses.html
[15] http://dl.aircrack-ng.org/breakingwepandwpa.pdf
[16] https://sviehb.files.wordpress.com/2011/12/viehboeck_wps.pdf
[17] http://www.aircrack-ng.org/doku.php?id=deauthentication

3 Local High-Speed Monitoring with Parallel NIDS

The previous chapter was devoted to the *prevention* of attacks on local area networks originating from internal attackers or targeted attacks. Some threats, e.g., external attacks, cannot easily be prevented because the external communication infrastructure is not under common control. Therefore *Network Intrusion Detection Systems* (NIDS) are often used as a *reactive measure* to *detect* attacks. NIDS have been applied with similar design and sensor placement principles in productive environments since the 1990s. Network technologies and domains, however, have been changed dramatically since then. Highly variable communication relations and constantly increasing network bandwidths more frequently force NIDS to handle high peak rates. This is illustrated in Figures 3.1 and 3.2. Figure 3.1 measures the maximum peak rates that have occurred within 5 minute intervals during a day within the data center backbone of our (small) university. Figure 3.2 depicts the same type of measurement for the file and VM services of the computer science institute. In the figure it can be seen that the peak rates often reach 1 Gbit/s, 2 Gbit/s are also possible. Single-threaded NIDS, however, were originally designed for a performance from 100 MBit/s to 500 MBit/s. Single-thread performance improvements for this type of system can often be achieved only by increasing the processor cache size.

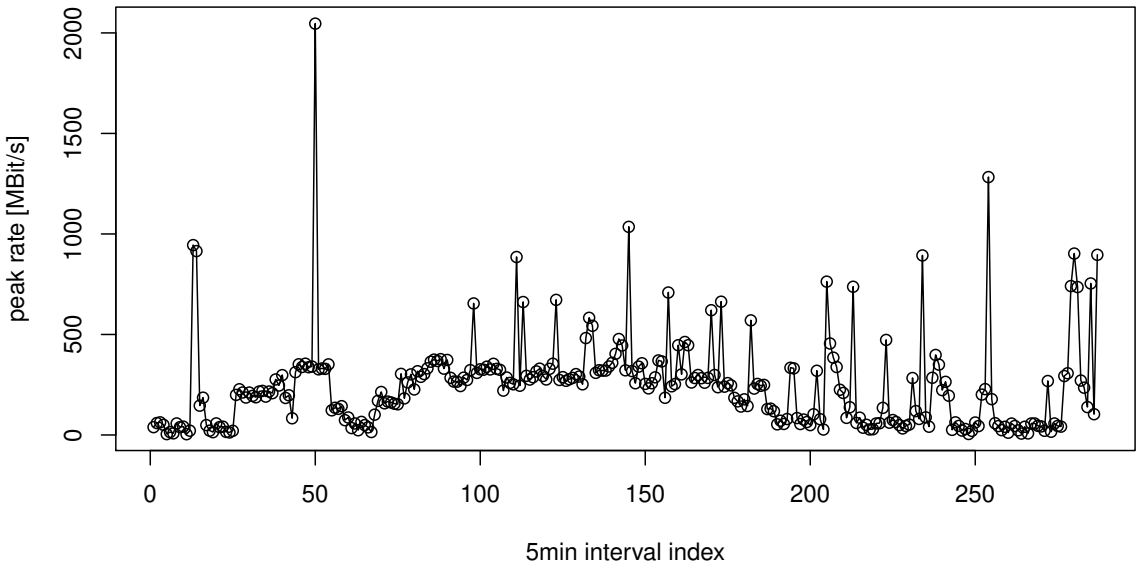

Figure 3.1: Peak rates for the most active day of the week in the university data center

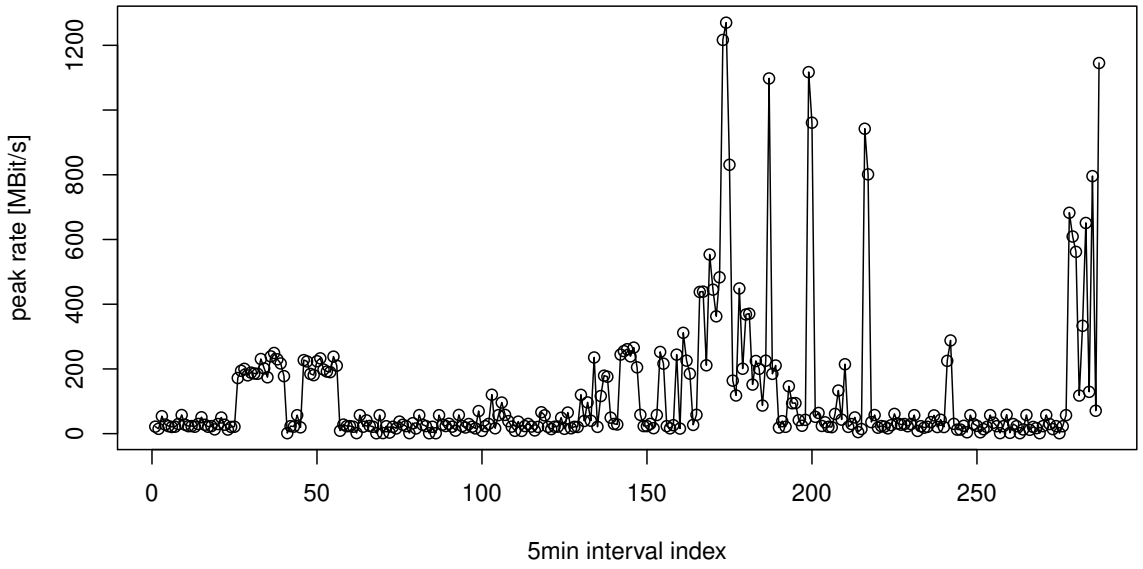

Figure 3.2: File and VM services peak rates for the most active day of the week in the university data center

Various approaches tried to optimize intrusion detection analyses through specialized hardware or optimized operating system kernels. Most of them favor parallelization to speed up the performance [73, 125, 126, 74, 127, 75, 128]. However, many of these approaches partially switch off essential parts of the IDS's analysis and detection capabilities when measuring the performance increase of their method. Thus, the evaluated configurations are not able to detect real-life attacks. In addition, they do not compare the performance gain through parallelization – independently of their focus on hardware or software solutions – with the theoretically achievable one. Recent investigations have also shown that parallel approaches often do not benefit from the cache sharing capabilities of modern multi-core CPUs [129]. They do not scale well regarding random access memory bandwidth shared among multi-threaded applications which requires a very efficient cache usage.

In this chapter, different approaches that address the caching and parallelization problem to optimize the system performance of NIDS are investigated. Based on a detailed performance profiling, it will be shown why other approaches fail to achieve the expected increase. As consequence, a novel NIDS analysis approach is being proposed that is capable of meeting the monitoring requirements of modern computer networks. The approach focusses on user-space solutions for non-distributed multi-core systems in real-world scenarios and does not make assumptions about the underlying operating system kernel. The performance gains are evaluated using a prototype which reacts to changes of network traffic characteristics in a very short time. In contrast to other approaches, the resulting performance gains are also compared with the theoretically achievable maximum. First reflections on the approach were published in [130].

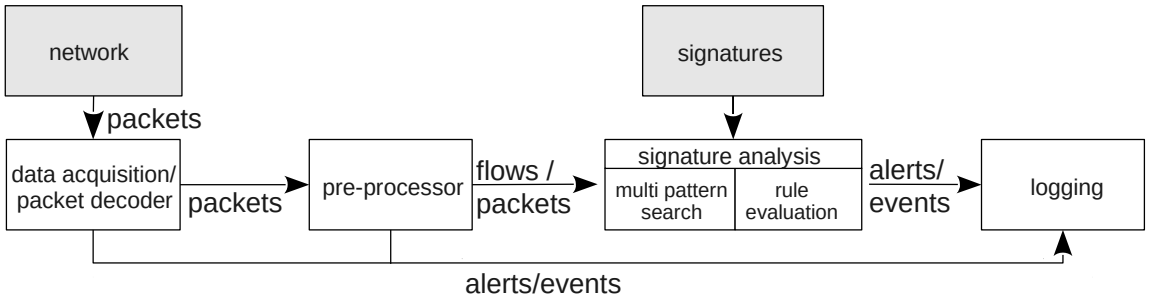

Figure 3.3: Pipeline architecture of SNORT

3.1 The Parallelization Approach of Suricata

The most popular parallelized NIDS is SURICATA[1] which is signature-compatible to the single-threaded NIDS SNORT[2]. This section introduces the two systems and discusses the practical results of SURICATA in parallelizing the NIDS analyses.

Snort & Suricata

The most widespread open-source NIDS is SNORT, which applies a *pipeline architecture* that completely analyzes each network packet in one step (see Fig. 3.3). Pipeline architectures implement a zero-copy strategy. Network packets are captured by a data acquisition module using a ring buffer. Packet processing is done in three stages: (1) packet decoding to extract protocol headers from network frames, (2) preprocessing of decoded data depending on the identified protocol (e.g., reassembly of TCP streams), and (3) packet and flow analysis in the detection engine applying IDS rules (multi-pattern search, rule evaluation). Detected attacks are logged and indicated. If the detection engine is not capable to keep up with the incoming network data, not yet analyzed packets in the ring buffer are overwritten by new ones.

The basic principle of SNORT has been parallelized in the IDS SURICATA (cf. Fig. 3.4), which basically executes the SNORT pipeline stages in separate threads. In addition, it parallelizes the detection stage as follows. SURICATA processes several packets in one thread and transfers them to other threads via multi-writer/multi-reader packet queues. All packets are allocated from a single global memory/packet pool. Network data is considered as a compound of multiple network flows. In the preprocessing stage network flows are statically balanced over the input queues of the various detection engines (the calculation of the destination queue is a simple modulus of the (UDP/TCP) source and destination ports which is used as an index for a queue table).

[1] http://www.openinfosecfoundation.org
[2] http://www.snort.org

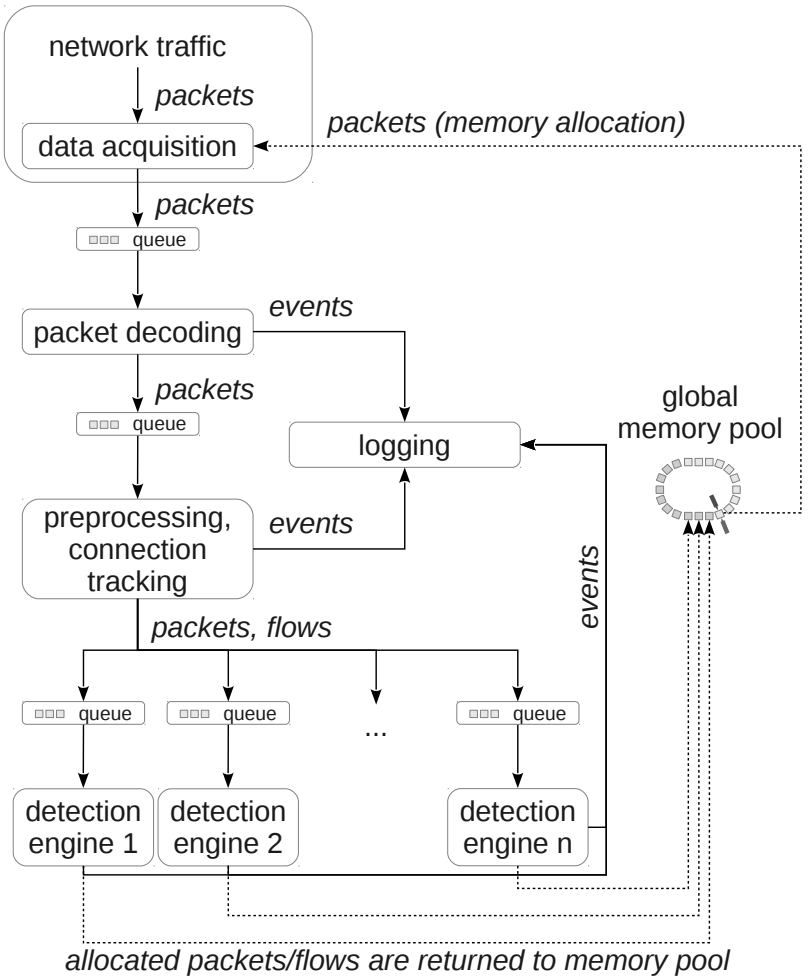

Figure 3.4: SURICATA's parallelization architecture

As soon as the analysis of one detection engine has been finished, the packet buffer has to be returned to the global memory/packet pool. Since various threads have to synchronize their access to this memory pool, this is a serious disadvantage for the system performance.

Practical Results versus Potential Performance Gains

This subsection compares the theoretically achievable performance gain through parallelization with the results achieved in practice by SURICATA.

Used datasets. For this purpose recent versions of SNORT (2.9.4.5) and SURICATA (1.4.1) were examined on an Intel Xeon machine (E5645, 6 cores) applying four

	nsa_p1	nsa_p2	nsa_p3	west_pt	defcon	acsac06	indust.
file size[MB]	4,768	4,768	4,294	726	5,723	6,452	2,389
packets[kpkts]	7,081	4,777	7,322	5,230	20,769	12,451	14,113
TCP[\%]	88.39	91.77	93.98	85.04	99.42	98.24	0.96
UDP[\%]	3.0	0.76	4.02	1.84	0.28	1.2	0.02
IPv4[\%]	92.0	92.64	98.53	98.87	0.02	99.52	0.1
IPv6[\%]	0.1	0.05	0.12	4e-04	99.86	0.0	0.0
Other[\%]	7.9	7.31	1.35	1.13	0.12	0.48	0.12
Profinet[\%]	0.0	0.0	0.0	0.0	0.0	0.0	98.9

Table 3.1: Characteristics of the used datasets

datasets from different sources. The first dataset (*nsa*, *west_pt*) comprises packet captures of the Cyber Defense Exercise 2009 (CDX[3]) with real attacks of the National Security Agency (NSA) for a test network of the West Point Military Academy. It contains a lot of reconaissance (mostly based on nessus[4]/OpenVAS[5]), e.g., scans for SSH, IMAP, VNC, SNMP, RPC, and Microsoft's internet information services (IIS) and terminal services. There are also attacks on various services, e.g., on DNS, IIS, and HP OpenView with buffer overflow and file access attempts. In addition, there are at least two denial-of-service (DOS) attacks included (UDP flood and DNS amplification test). The second data set (*defcon*) contains attacks that have been captured at the conference DEFCON 2012. It is based on crafted services on top of IPv6/TCPv6. This data set is a good candidate for generic detection methods, e.g., shell code detection signatures and machine-learning. The third dataset (*acsac06*) involves a set of attacks for different target platforms which have been published in [131]. It covers 124 vulnerabilities with different attack variants (420 attack instances) and obfuscations (3549 attack instances). Attacks that are detected by Snort are reconnaissance (RPC, IIS scans), file access, bind of cmd.exe to the network, some worm propagation, and a lot of generic shellcode (various no-op-slides). Furthermore, a fourth dataset (*indust.*) was captured by our research group in a large industrial site. It does not contain any attacks and it is just an example for Ethernet-specific traffic. The deployed intrusion detection signature sets for the two IDS are the official rule set[6] for SNORT (2013-02) and the emerging threats rule set[7] (2013-03). For a fair evaluation, (1) several consecutive captures were combined to sufficiently large datasets (roughly 5 GB for each set), (2) all sets were preloaded into the RAM to prevent wrong measurements caused by input/output waitings, and (3) the (potentially biased) official rule set for SNORT was combined with the the emerging rule set (which is potentially biased with respect

[3]http://www.westpoint.edu/crc/SitePages/DataSets.aspx
[4]http://www.tenable.com/products/nessus-vulnerability-scanner
[5]http://www.openvas.org/
[6]http://www.snort.org
[7]http://www.emergingthreats.net

to SURICATA's internals) to a single rule set which is applicable for the two NIDS. The traffic characteristics of the analyzed datasets are listed in Table 3.1. For the sake of brevity, the table lists only the ratio of the major protocols.

Performance achieved in practice through parallelization. Table 3.2 lists the mean runtime of five runs of each IDS applied to each dataset and the resulting parallelization gain (*speedup*) of SURICATA compared to SNORT. Two of the datasets are not analyzable by SURICATA because of synchronization problems (livelock, segmentation fault). Obviously, SURICATA outperforms SNORT only for the analysis of the *defcon* dataset, while it is slower in analyzing the other captures. This will be explained in detail after comparing SURICATA's results with potential parallelization results for SNORT.

	Snort			Suricata	
	Snort [s]	Snort [Mbit/s]	Snort [kpkt/s]	Suricata [s]	Speedup
nsa_p1	54.8	729.9	129.2	218.2	0.25
nsa_p2	49.4	809.6	96.7	228.2	0.22
nsa_p3	63.4	568.1	115.5	152.6	0.42
west_pt	20.4	298.5	256.4	livelock	livelock
defcon	89.6	535.8	231.8	42.4	2.11
acsac06	209.6	258.2	59.4	segfault	segfault
indust.	6.2	3,232.3	2,276.3	21.8	0.28

Table 3.2: Runtime of SNORT and SURICATA

Theoretically Possible Speedup. For an independent consideration of parallelization gains among existing approaches and the later proposed approach, it is necessary to determine the theoretical acceleration of the NIDS analysis speed when parallelizing certain analysis steps. For this purpose, the runtime of the individual pipeline stages of SNORT were measured with its internal microbenchmark mechanism as fraction of the total runtime from Table 3.2. Table 3.3 contains the results. According to Amdahl's law [132] the achievable speedup s by parallelizing a program into n units is:

$$s = \frac{1}{r_s + \frac{r_p}{n}} \; with \; r_p = 1 - r_s, \; r_s \in \mathbb{R}, \; 0 \leq r_s \leq 1 \tag{3.1}$$

where r_s represents the serial part of the program and r_p the parallelizable parts. Both, r_s and r_p are expected to be normalized to the interval $[0.0, 1.0]$ in this formula. SURICATA decouples the packet decoding and preprocessing stages from the detection stage (*component-based parallelism*). The detection stage is further parallelized by concurrently analyzing several network packets/streams (*data-based parallelism*). The data acquisition stage is the only phase that directly depends on the incoming packet stream and is therefore serial.

	other + data acquisition [%]	packet decoding [%]	preprocessing [%]	detection [%]	logging [%]
nsa_p1	10.78	6.02	10.33	72.24	0.63
nsa_p2	10.17	5.49	8.86	75.04	0.44
nsa_p3	10.64	5.55	10.52	72.74	0.55
west_pt	10.86	7.18	46.06	34.18	1.72
defcon	7.42	8.45	39.01	43.88	1.24
acsac06	19.82	2.67	13.80	63.25	0.46
indust.	70.07	15.30	3.86	3.25	7.52

Table 3.3: Percentage of analysis time for each stage of the SNORT pipeline

Based on the configuration of SURICATA, it is possible to calculate the maximum acceleration for three theoretically parallel variants of SNORT. The first variant is *component- and data-parallel (amdahl)*. The acceleration can be calculated by assigning the data acquisition stage to r_s and all other stages to r_p in Amdahl's formula. The *nsa_p1* dataset, for example, has an analysis effort of roughly 10% ($r_s = 0.1078$) for the serial program part and 90% ($r_p = 1 - r_s = 0.8922$) for the parallelizable part. In this case, the expected performance gain (*amdahl*) on a 6-core CPU is 3.9. However, some components/stages in the SNORT pipeline might not scale as expected in the parallel case. The preprocessing stage, for example, sometimes has to aggregate input data (e.g., for TCP reassembly and port scan detection) and is therefore serial in some cases. Thus, it is useful to calculate a second *data-parallel only (amdahl_dp)* variant of SNORT by assigning the detection stage to r_p and all other stages to r_s. In the most extreme case, the input data do not provide any data parallelism at all for the analysis. Therefore, it is necessary to calculate a third prediction for a *component-parallel-only (comp_par)* variant of SNORT based on the serial data acquisition stage and the most time-consuming component for each dataset

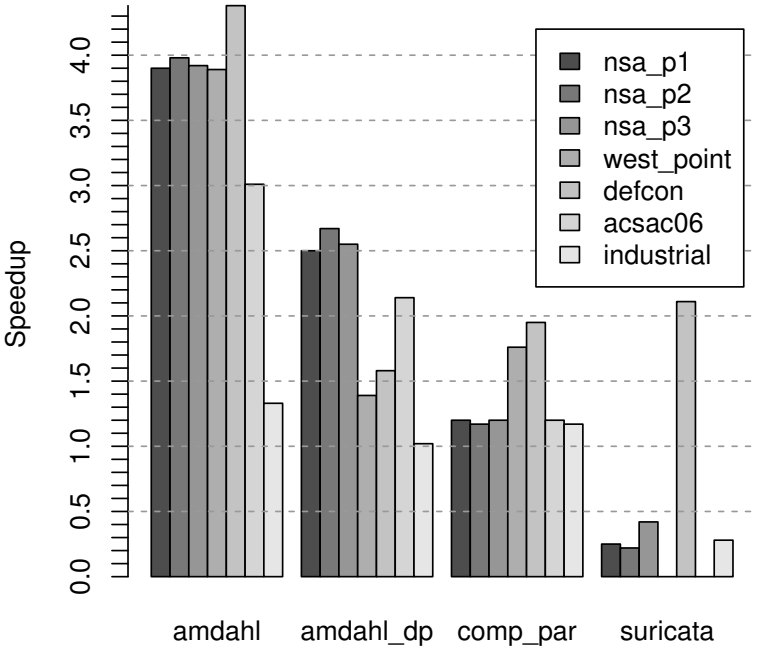

Figure 3.5: SURICATA's speedup versus the prediction using Amdahl's formula for SNORT

(e.g., $\frac{1}{r_s+detection}$ for the *NSA* datasets and $\frac{1}{r_s+preprocessing}$ for the *west_point* dataset – assuming that there are enough processor cores available for the other pipeline stages). Figure 3.5 compares the results of these calculations for 6 cores ($n = 6$) with the achieved speedup of SURICATA (cf. column *speedup* in Table 3.2). Note that this comparison is legal because the pattern search code of SURICATA (Aho-Corasick algorithm), which constitutes most of the analysis effort of the parallelizable program part (*detection stage*), is a copy of SNORT's code. As it can be seen from Figure 3.5, there is a noticeable gap between the theoretically possible parallelization and the results achieved by SURICATA.

Limits of Suricata's Parallelization Strategy. The basic problem of SURICATA and other IDS approaches (e.g., [75, 74]) which apply parallelization is that the proposed architectures often ignore the CPU cache, memory access patterns, context-switching problems, and busy-waiting. In SURICATA, the problems are most likely caused by implicit synchronization, excessive locking, and bad CPU cache usage. First, the data acquisition and the detection engines are implicitly synchronized with each other via the global memory pool because packet buffers have to be returned after finishing the analysis. This may lead to massive contention when accessing the memory pool depending on the number of detection engines, which directly relates to the number of processor cores. According to recent investigations [133] this has a much higher impact than intuitively thought, because the access to critical sections must be modeled as another serial part of the program in enhancements to Amdahl's formula. Furthermore, SURICATA *statically* balances network flows over several analysis units in the *detection stage* which may lead to some idle detection engines, while others are still processing incoming flows due to differences in packet processing times.

Additionally, it is necessary to understand the impact of *single* packets for the IDS analysis. If we divide the runtime of the single-threaded IDS SNORT for each dataset in Table 3.2 by the number of packets of each dataset (cf. Table 3.1) the results for most datasets are roughly between 5 and 15 μs. A *microsecond* execution time is *lower by an order of magnitude* compared to the usual operating system time-slice for kernel-level threads (e.g., around 20 *ms* for the CFS scheduler of linux). It makes little sense with respect to the operating system overhead (context-switching) to construct an IDS architecture which reacts to every single packet. SURICATA processes *each* packet individually on a *per-function* basis despite of the queues between its modules. This can cause side effects regarding the CPU caches that are depicted in Figure 3.6. First, the data acquisition stage of SURICATA stores the network packets in the order of their arrival into the global memory pool and then it stores pointers to the packet buffers into the queue of the packet decoding stage. Further stages access the queued packet pointers in the arrival order until they reach the preprocessing and connection tracking stage. Next, the packet pointers are statically balanced into the queues of the different detection engines based on the underlying network flows. At this point,

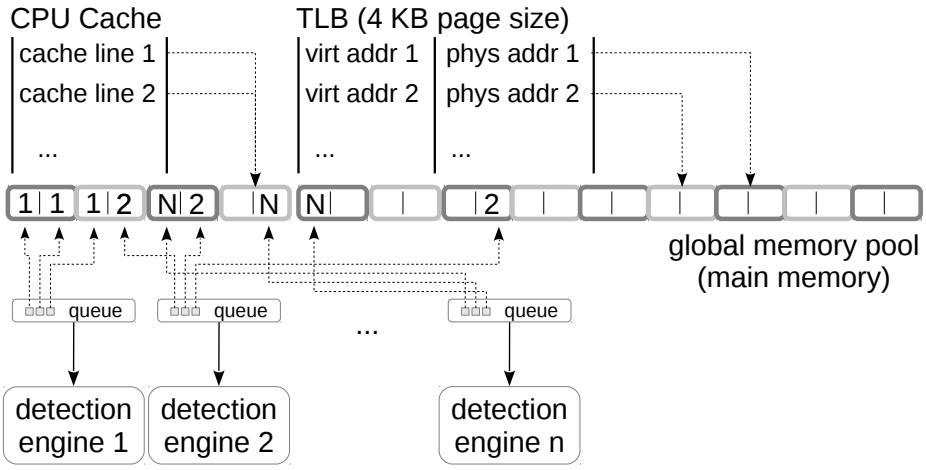

Figure 3.6: Relation between the CPU caches, the global memory pool, and the packet queues in SURICATA

the memory access becomes random because the packet pointers in the queues of the different detection engines point to *interleaved* network flows. The lock-based access to the synchronization variables of the detection engines queues can cause additional randomizations by context-switches between the connection tracking stage and other detection threads at any time. The CPU has to manage two types of caches which are affected by this behavior. The first one is the L1/L2/L3 cache hierarchy which can store parts of the packet data from the main memory and other IDS data structures, if used correctly. This cache hierarchy refers to the *physical memory addresses* of the buffered data. SURICATA tries to mitigate impacts on the cache hierarchy by allocating the memory pool with a sufficiently small number of packet buffers that fit into the L2 cache of most systems. However, the CPU also manages a second cache which is called *translation lookaside buffer* (TLB). This second cache maps *virtual memory addresses*, e.g., the pointers to the packet pool, to *physical memory addresses*. The set of the buffered memory translations is usually very small and a cache miss in the TLB is nearly as expensive as a miss in the other CPU caches. Furthermore, the TLB can become invalid in the case of a context switch[8] between different processes. The combination of single-packet processing with an execution time below the OS time-slice for threads and the random access of different detection engines to interleaved memory regions can cause cache misses at the line rate of the incoming packet stream.

To verify the assumption regarding the context switches and cache misses, SNORT and SURICATA were profiled with the Linux *perf* toolkit[9]. As can be seen from Table 3.4, the context switches (*co sw*) and TLB misses of SURICATA exceed indeed that of SNORT by several orders of magnitude.

[8]context switch: stores/restores the state of a process (which involves a change of the address space)
[9]https://perf.wiki.kernel.org/index.php/Tutorial

	Snort			Suricata		
	co sw	cache miss	TLB miss	co sw	cache miss	TLB miss
nsa_p1	617	41,834,414	83,059,435	3,623,893	551,600,482	670,692,478
nsa_p2	617	32,438,877	63,476,800	3,729,426	349,591,663	638,931,821
nsa_p3	628	56,720,450	90,978,462	2,719,206	515,808,743	724,476,278
defcon	760	134,356,466	132,972,654	1,903,271	344,009,157	602,544,996

Table 3.4: Context switches and cache misses for Snort and Suricata

3.2 Further NIDS Optimization/Parallelization Approaches

Various approaches have been published to speed up intrusion detection systems and to improve their analysis capabilities. The majority of published papers deals with Snort because of its large signature base and its far-reaching acceptance. For the sake of brevity, only the major results are outlined here.

Input reduction and optimization aims at buffering, filtering, and/or reordering of network packets before they enter the intrusion detection system. The optimization approaches follow the course of a packet through the network card, via the operating system kernel, to the userspace (applications/NIDS). Xinidis et al. [125] try to optimize the IDS input on the network card by applying early packet filtering (leading to a performance improvement by 8%) and by ordering of packets according to their destination ports to improve the cache locality of the IDS. This results in a recurring application of equal IDS rules on the packet streams inducing a good cache-hit rate with a performance improvement of 10–18%. In addition, a *static* load-balancer has been proposed which distributes the data flows within the network card to various sensors (input buffers). However, static load-balancing may cause imbalances. Therefore, load-balancing should run on the general-purpose CPUs because this enables *dynamic* distribution concepts (see Sections 3.4 and 3.5). The next stop of a packet on the route to the NIDS is the operating system kernel, which encapsulates the access to the network card. In [134], Fusco and Deri analyze bottlenecks of the Linux network interface card (NIC) drivers that aggregate all packet queues of modern network interface cards to a single queue for interfacing with the user space. They provide a special driver and a user-space API offering the possibility to directly attach user-space programs to the queues of the network interface card to circumvent the (to a certain degree even redundant) OS-side processing.

Content analysis optimization additionally considers the packet payload processing. In [73], the content of network packets is split into overlapping fragments which are analyzed by multiple independent processing units on a network card. Another approach [74] moves the network packets to a graphics processor (GPU) to evaluate packets or

packet fragments in parallel and in a later work [135], the same authors combine this method with a load-balancing network card on the input side. These approaches are discussed in more detail in the next two sections. In [136] Yang et al. try to boost the evaluation of regular expressions used in many IDS during content analysis by replacing deterministic finite state automata through parallel nondeterministic ones. Another approach of Smith et al. [76] suggests an automaton for regular expressions which reduces the state space by augmenting traditional finite state automata with a scratch memory for small, but highly efficient computations (e.g., counting). However, SNORT rules, for instance, are usually accompanied by static search patterns which are evaluated first. The evaluation of regular expressions is therefore skipped in most cases, i.e., only marginal performance gains are possible with these approaches. During the experiments for this work the regular expression evaluation in SNORT never achieved a share of more than 0.2% of the total analysis time.

3.3 Multi-threaded NIDS under Attack Conditions – Discussion of Related Work

Many of the existing parallel approaches try to reduce the input into the NIDS through pre-filtering of packets and to avoid synchronization problems between various IDS threads by turning off pipeline stages which has far-reaching consequences for the detection capabilities of the systems. This section discusses these consequences for three existing parallelization approaches. The GNORT approach of Vasiliadis et al. [74] switches off the preprocessing, the rule evaluation, and the logging (see Fig. 3.7). This essentially leads to a kind of network grep[10], but not to a functional NIDS. Later the same authors have developed a system, called MIDEA [135], that enables at least the preprocessing stage, but continued to disable the rule processing and the logging. This expands the capabilities of the system from a pure packet-based analysis to a flow-based analysis. The load-balancing approach presented in the publication, however, has the problem that it cannot distribute bidirectional flows (bi-flows). Thus, the analysis of attacks in which both communication directions are related to each other (e.g., the client request has to be correlated with the server response to detect the attack) is not possible. Another approach by Jamshed et al. – KARGUS [128] – disables the pre-processing stage and applies data parallelism on a packet level. Since this approach does not allow analyzing flows, it essentially disables the rule evaluation with respect to the flow state. This means, for example, that search patterns which span multiple packets cannot be detected in the multi-pattern search. In addition, attacks that require a correlation of multiple data flows among themselves, e.g., port scans during the reconnaissance, are not recognized in the rule evaluation.

[10] Unix command that searches input files for lines containing a match to the given pattern

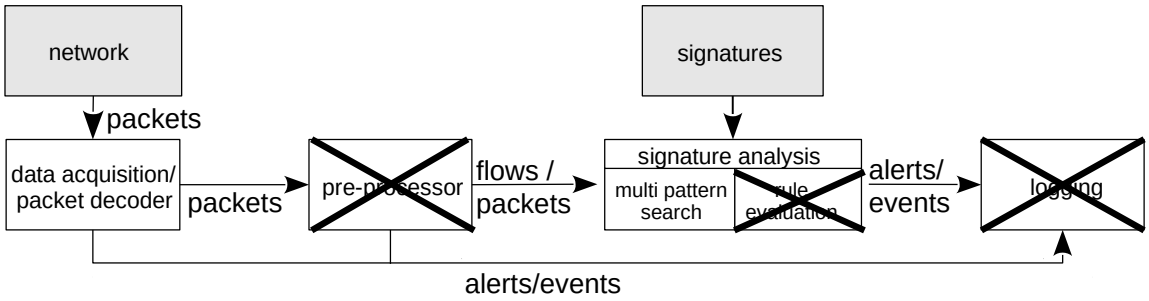

Figure 3.7: Typically removed pipeline stages in current parallel NIDS approaches

Moreover, the way in which parallel NIDS systems are evaluated is questionable. Turning off pipeline stages, for example, shifts the baseline of the experiments, which is not discussed in any of these papers. This is illustrated in Table 3.5. As can be seen in this table, turning off the preprocessor for the flow analysis speeds up the performance between 50% and 300%. Therefore, it is unclear whether the performance of previous parallelization approaches (which are unfortunately often not specified as performance increases over a clear baseline) result from the parallelization or from the shut-down of pipeline stages. The table also shows how extremely the shut-down of the first pipeline stage influences the detection rate of SNORT. It detected only between 19 and 24% of the attacks.

	flows enabled		flows disabled			detected
	MBit/s	kpkt/s	MBit/s	kpkt/s	speedup	attacks
west_pt	298.5	256.4	428.9	368.3	1.44	24.1%
acsac06	258.2	59.4	771.0	177.4	2.99	19.4%

Table 3.5: Performance versus accuracy of SNORT with/without flow analysis

Further questionable assumptions of former approaches are the distribution of normal and attack data in traffic: (1) Typically it is assumed that more than 99% of the network traffic are normal data which do not contain any attacks. NIDS are optimized for this case and are significantly less efficient if this distribution changes. (2) In the understanding of these approaches, an attack is often based on one attacker, one vulnerability, and one target. Sometimes multiple targets are attacked, but the limitation to one attacker and one vulnerability often persists. Real attack behavior is, however, different. Figure 3.8 represents different traffic situations that may occur during an attack. The NSA data sets used in the Section 3.1 have already shown that an attacker usually does not exclusively attack a single system, but also executes DOS attacks for distraction purposes (left side of the figure). Traffic patterns that are found during a DOS attack possess an 1:n relationship, e.g., one attacker, one host, n connections for SYN flooding, or one attacker and many hosts during DOS attacks on multiple hosts. Other DOS attack options are reflection attacks with an n:1 relationship, e.g., during a SYN flooding attack with a forged sender address, in which the answers are sent to

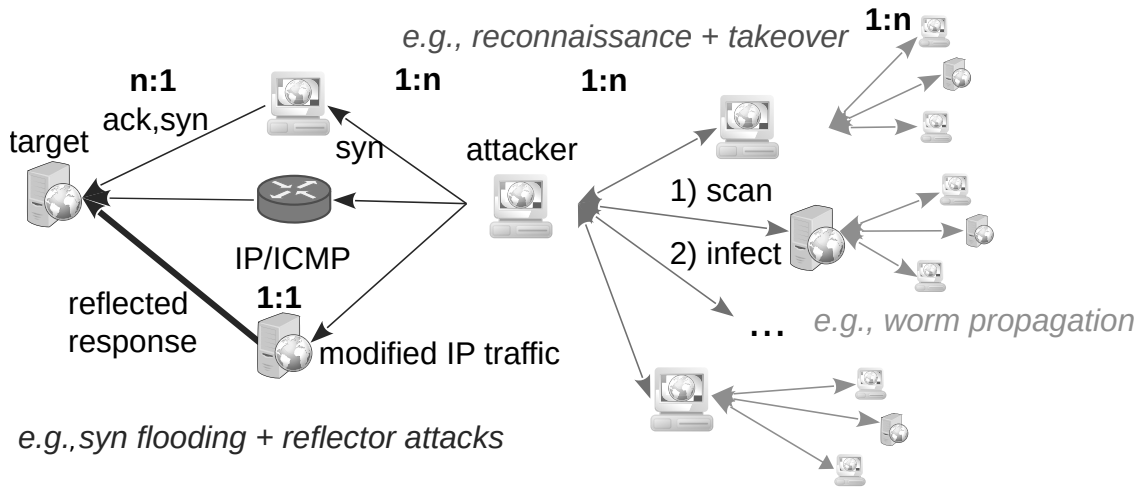

Figure 3.8: Reconnaissance, takover, and DOS behavior

the destination of the faked address (upper left part of the figure). Some reflection attacks have a 1:1 relationship, i.e., the attacker sends small packets to a server by means of a faked source address. The typically larger response of the server compared to the input is then sent to the faked address (lower left part of the figure). At the same time, the attacker performs his/her targeted attacks (right side of the figure). In the first phase (reconnaissance) this involves massive scans of the target network with similar traffic characteristics as in DOS attacks. After that there are usually targeted infections (takeover) of individual hosts, which are sometimes lost in the noise of the DOS attacks. If the transmitted malicious software has its own distribution routines, another harmful traffic emanates from each infected host, e.g., as observed during worm propagation. All of these traffic patterns typically produce a massively larger traffic volume, as assumed in former analyses of parallel NIDSs. Unfortunately, NIDSs are not optimized for this kind of traffic and respond with a huge decrease in analysis efficiency.

3.4 Design Options for Fully Functional Parallel NIDS Architectures

For each parallelization approach, it is important to determine in advance the components, functions, and algorithms that are crucial for the computing time. The micro-benchmark mechanism of Snort used in Section 3.1 has provided first indications for this (cf. Table 3.3). It identified the detection engine as the most compute-intensive component. In addition, this component can be parallelized in a safe manner without any major impact on the other pipeline stages which is necessary for a fully func-

tional parallel NIDS variant. There are essentially two different approaches for the parallelization of individual components – the *component parallelism* and the already mentioned *data parallelism*.

Component or Function Parallelism.[11] Many programs have independent program parts that can be executed in parallel on the *same data*. Program parts are individual source blocks or function calls – that's why this type of parallelism is also known as function parallelism. The component parallelization assigns these blocks or functions to individual lightweight processes (*threads*) or heavyweight processes (*process/program*). Examples of this type of parallelism are the statistical analysis of a packet in the context of port scans with a simultaneous comparison of its contents using NIDS rules or the reassembly of multiple TCP segments with a simultaneous signature analysis of the reassembled data stream. Such parallelization would involve the separation of the individual pipeline stages of the NIDS into several parallel sections, whose simultaneous execution, however, leads either to a myriad of parallel activities or whose mapping to a few parallel control flows has a high synchronization overhead. Another variant would be to aggregate smaller TCP segments to reasonable sizes (e.g., 64 KB) and to run the subsequent pipeline steps in parallel on already aggregated segments – which leads to a function parallel analysis on the same data stream.

Data Parallelism. In this parallelization variant the *same function* is applied in parallel on different data sets. The *single-instruction/multiple-data* (SIMD) principle performs the same instruction sequence on different parts of the data in parallel on different processor cores. It is a hardware architecture principle for parallel computers that is not really suitable for parallelization in the context of NIDS. Single-program/multiple-data (SPMD) extends this principle on identical programs/program parts that are executed asynchronously on different data. Relevant program parts for this principle are in particular loops that iterate over data fields to perform calculations on it. In the NIDS area, such calculations occur, for instance, in the application of string-search automata on different data flows inside of the detection engine.

Application of Function and Data Parallelism in NIDS

Function and data parallelism can take place on a variety of levels in NIDSs. Therefore, a general model of a parallel NIDS is introduced, based on which various fine-grained parallelization variants can be discussed. In general, the *function* of a signature-based NIDS is *to match a set of intrusion detection rules* with the incoming *set of packet/flow*

[11] The terms component and function parallelism are used interchangeably in the literature. The former seems to refer to architectural separable components, while the latter often refers to dividing of algorithms. Subsequently, the term function parallelism is used for both terms.

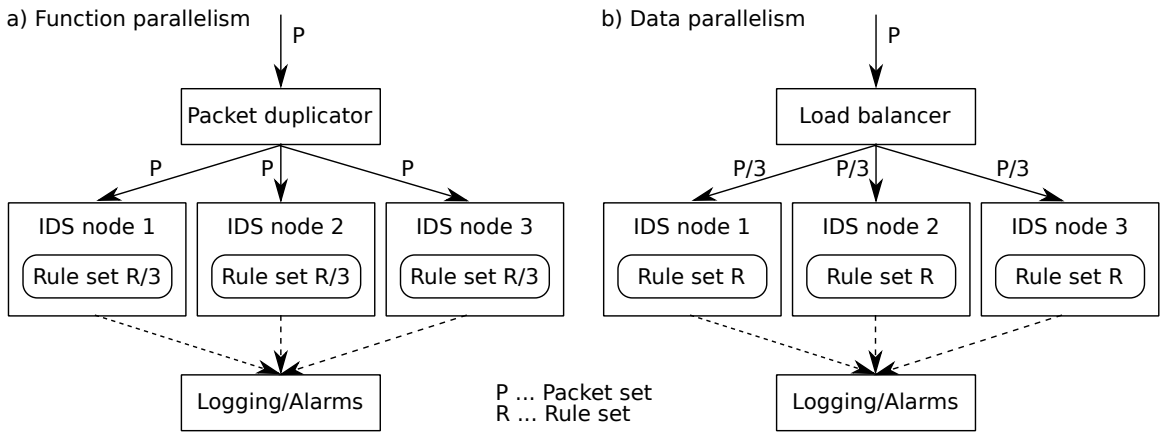

Figure 3.9: Example of two general parallel NIDS architectures

data, and to trigger an alarm, or to log the incident if there is a positive match of a rule with a packet/flow. Figure 3.9 depicts two possible parallel NIDS architectures that result from this general consideration. Function parallelism can be achieved by duplicating packets at the entrance to the rule processing functions and by breaking down the rule-based logic of the IDS into several disjoint calculations or subsets (cf. Fig. 3.9a). Data parallelism can be achieved by distributing packets/flows with a load balancer to different IDS nodes with identical rule processing logic (cf. Fig. 3.9b).

Application of Function Parallelism

The splitting of the control logic for the rule-processing can be done in two ways: (1) either the entire IDS is considered as a black box and various instances are configured with different parts of the rule set or (2) the IDS rule logic is split internally into several parts. Subsequently, the results of some experiments are presented which are based on the most coarse-grained and most fine-grained function parallel approach for SNORT.

Coarse-grained Function Parallelism. A coarse-grained function parallel variant can be implemented relatively simply. In an experiment, the 52 rule groups of the official SNORT signature base were aggregated into eight groups with approximately the same computing time. These eight groups were further distributed according to the function-parallel architecture in Figure 3.9a on three IDS nodes, which yields 966 possible distributions according to the formula for the Stirling numbers of second kind[12]. For 75% of these distributions, a performance improvement just over 20% could be achieved.

[12]http://oeis.org/A008277

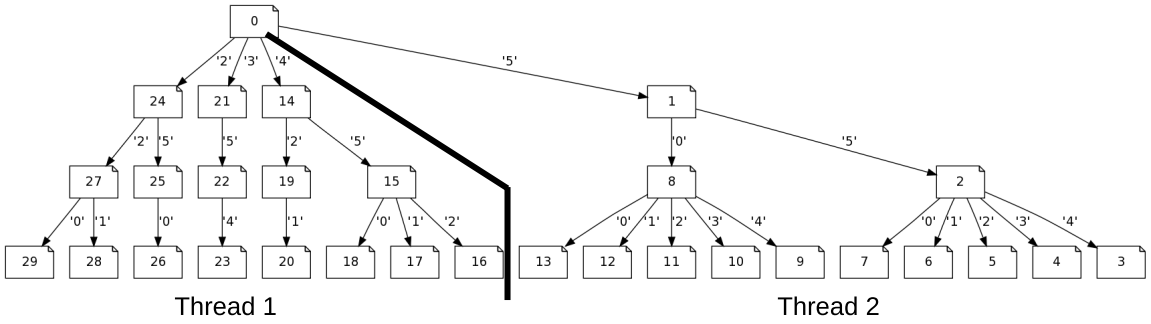

Figure 3.10: Parallel multi pattern search

Fine-grained Function Parallelism. The most fine-grained function parallelism is the distribution of the static search patterns (character and binary strings) of a single IDS rule group across multiple processor cores. The approach of SNORT to the simultaneous search for multiple search patterns is based on an algorithm by Alfred V. Aho and Margaret J. Corasick [137]. The Aho-Corasick algorithm constructs a search automaton out of multiple search patterns whose search costs are linearly dependent on the length of the text to be searched (*O(n)*, n ... length of input text). The construction of the automata is carried out in two phases. In a first pre-processing step, the search patterns are stored in a search pattern tree (*Trie*). The second phase transfers the Trie into a deterministic search automaton. Figure 3.10 represents a possible parallelization approach for the search of character strings based on an example Trie of a SNORT rule group. The parallelization is performed by breaking down several search patterns in various data structures, e.g., by splitting the Trie into two halves. Based on the divided data structures, several search automata are constructed with the *Aho-Corasick* algorithm that can be executed in parallel at runtime. An experiment with a parallel prototype of an old version of SNORT (2.8.6) achieved a performance increase between 12.6% (*nsa_p3*) and 25.7% (*nsa_p1*) for the data sets from Table 3.1. In a later version of SNORT (2.9), however, the search automatons were split into a fast automaton for the most-distinguishing search patterns of all rules in a group and a slow automaton for the other search patterns, which led to a 97% reduction in their size, so that the new serial version of SNORT outperforms the prototypical parallel variant.

Application of Data Parallelism

Just as in the case of function parallelism, data parallelism can be implemented as a coarse-grained or fine-grained variant: (1) In the first case, the IDS is again considered as a black box and a load balancer distributes various data streams on different instances of the IDS, or (2) the packet processing logic is split internally.

Coarse-grained Data Parallelism. For the coarse-granular approach, a prototypical load balancer was implemented similar to the first pipeline stages of SNORT. The load balancer reads, analogously to the data acquisition stage of SNORT, the data packets from a network interface or from a file and identifies, as in the connection tracking stage, the contained TCP flows and UDP flows based on the 5-tuple *(protocol number*, *source address*, *destination address*, *source port*, *destination port*). The packets are written by the actual load distribution filter into several first-in-first-out (fifo) queues which are connected to SNORT instances. Packets are distributed as *bidirectional flows (bi-flows)*, i.e., the packets of the two communication directions of a connection are assigned to the same IDS instance. In an experiment with the data sets of Table 3.1, this load-balancer achieved a performance increase between 0.05% (*nsa_p2*) and 83.3% (*west_pt*). The unchanged runtime in the tests with the load balancer and the *nsa_p2* data set goes back to an unfavorable distribution of connections on the SNORT instances. Most of the parallel instances were idle during the measurements – a disadvantage which could be countered, e.g., by fine-grained data parallelism.

Fine-grained Data Parallelism. The most fine-grained data-parallel NIDS is the SNORT variant of Yu et al. [73]. It disassembles individual packets into packet fragments which are then analyzed in parallel on a network card. For an experiment, a prototype was implemented that mimics the approach of Yu, but which aims at parallelizing the IDS analysis using general-purpose hardware instead of assigning packets or packet fragments to special processing units (e.g., network cards or graphic cards). The performance results of this prototype ranged from performance losses of 8.4% (*west_pt*) to performance increases of 8.75% (*nsa_p2*). The performance degradation of the *west_pt* data set goes back to the fact that the average packet size for this data set compared to the overhead of parallelization is too low.

Conclusions From These Experiments

The performance improvement of just over 20% for the coarse-grained function parallel prototype was quite consistent for most of the signature distributions. These results are encouraging, but their consistency over a broad range of signature distributions indicates at the same time that further performance increases are no longer possible. The fine-grained function parallel approach is less performant than newer serially executable variants of SNORT. Therefore, it is recommended to follow the data-parallel approach.

The comparison of the coarse-granular and fine-granular data-parallel variants shows that each of these approaches has its own difficulties depending on the input data. The problems associated with the coarse-granular distribution of the load comes from the fact that large data flows (so-called *elephant flows*) disproportionately occupy individual IDS instances, while the other ones are idle. In contrast, the fine-granular

parallelization approach has problems with data flows of too small packet sizes (so-called *mice flows*). Therefore, a variant is required that combines the two methods and which always works optimally when it is faced with changing traffic characteristics.

3.5 A Novel Dynamic Parallelization Approach for NIDS

Based on the preceding discussions of the Sections 3.1 and 3.4 this section presents an approach that is capable to cope with the increasing network dynamics (see Figure 3.11). The principle idea is that batches of packets are passed through the IDS from module to module in a quasi-synchronous manner from the data acquisition to the analysis. The approach consists of three stages. In the first stage, the *data acquisition*, packets are captured at the local area network interfaces and stored in the respective packet pools. In contrast to Suricata, the packet pools are local to the respective network interface, and allocated packet buffers are never explicitly be returned to the pool. The data acquisition modules form the *batches*, which may consist of packets or arbitrary events. Furthermore, they ensure that the amount of memory used for storing all batches is below the size of the CPU cache. Then the batches are pushed forward to *intermediate preprocessing modules*, e.g., packet filters, which belong to the second stage. Since there are no queues between the modules, a batch is only forwarded between two modules when the processing of the previous one has been completed, i.e., the processing in the first module is blocked if the second module still processes a batch. The packet filters perform some preprocessing and preliminary analyses to reduce the incoming packet stream. Thereafter, the batches are forwarded to the *load balancer* and in some cases, the batch is forwarded in parallel to a flow exporter during this stage. The load balancer assigns the batches to the *detection engines*, the third stage. Here, all detection engines formally get the same batch, but the load balancer assigns different ranges of packets to be analyzed depending on the number of CPU cores or IDS instances. The same thing happens with the detection engine behind the flow-exporter, but this engine receives only a part of the data, e.g., the protocol headers with just a small amount of payload. After finishing the analysis the packets are discarded. There is no need to return the buffers to the packet pool. The modules are implemented by threads. By *pushing* batches with enough analysis effort for a full thread-execution time-slice instead of pushing single packets from module to module, i.e., from thread to thread, the thread-activation scheme of the operating system is forced to essentially follow the packet flow through the IDS (in contrast to the random activation of the pull-based scheme of Suricata). Therefore, this concept represents a *push-based approach*.

The concept differs from other approaches by the following characteristics, which, where necessary, will be explained in more detail afterwards: (1) packet batching in-

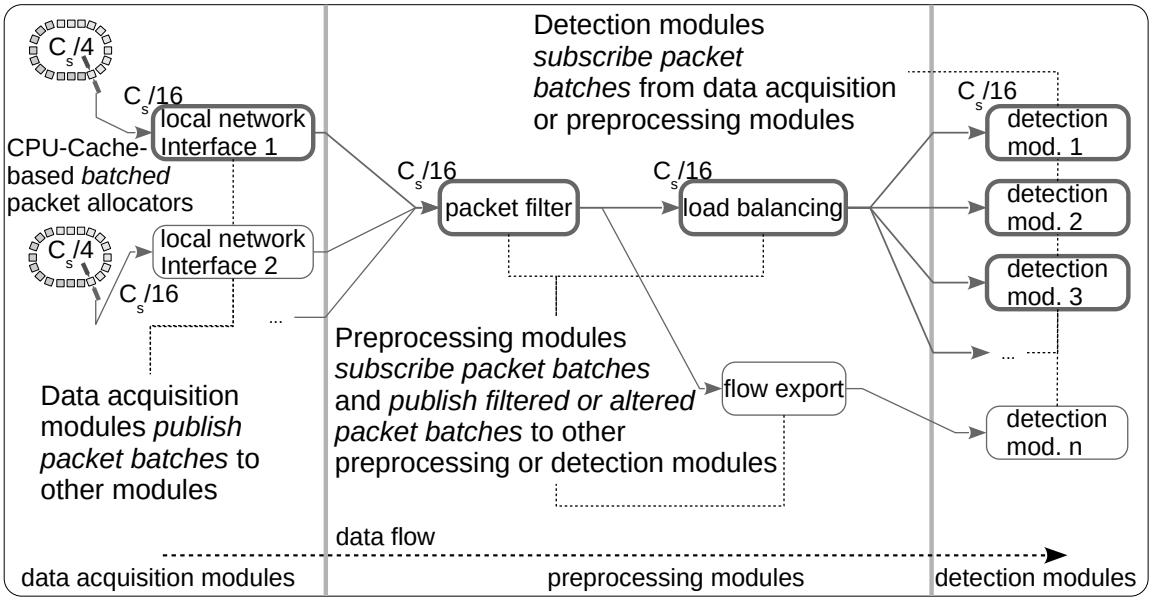

Figure 3.11: Push-based dynamic parallel NIDS approach

stead of single-packet processing increases the amount of processing inside a module and decreases the number of locking events and thread context switches, (2) the application of a CPU-cache-aware packet allocator for forming the batches never exceeds a configurable proportion of the CPU cache size, (3) the release of packet buffers at the sources (e.g, by the data acquisition or preprocessing modules) avoids the implicit synchronization of threads for memory allocation, (4) the thread activation scheme essentially follows the packet flows and, thus, increases the probability of cache hits, and (5) the possibility to precisely measure bottlenecks on the output path of each module by measuring blockages in the quasi-synchronous execution chain can be used for dynamic load balancing capabilities and adaptations of the packet batches.

Determination of Needed Parallelism and Memory Management

The parallelism of this approach is constrained by the following three assumptions. (1) Usually the packet sources are the only modules which allocate or release packet buffers. A module that allocates new packet buffers (e.g., a data acquisition module or a preprocessing module for packet reassembly) has to release them, too. (2) All modules process the packet batches one by one. (3) When a module has to forward a packet batch to directly attached modules, it waits until all of them are ready for processing (similar to a *dynamic barrier*). Based on these conditions, the packet sources (usually data acquisition modules) can calculate the maximum number of packet batches that can be analyzed in parallel. If we consider the model of Figure 3.11 as a directed acyclic graph, this maximum is equal to the longest chain of nodes

from a packet source to a packet sink in the sub-graph/tree to the right of the node representing the packet source. Each packet source pre-allocates p packet buffers for all packet batches based on the following formula:

$$p = \frac{s_c}{c * s_p} \tag{3.2}$$

in which s_c represents the size of the cache, c the number of cache partitions, and s_p the size of one packet. The concept of cache partitions is used to set aside memory, i.e., cache lines for further memory blocks which can be used for other utilizations of the CPU cache, e.g., for multiple packet sources and to ensure a good cache-hit rate for the pattern-search automata of the IDS analysis.

The big advantage of these calculations is that the exact knowledge of the maximum of parallel processed batches allows an automatic memory management with little synchronization points. This is illustrated for an exemplary setup of the architecture in Figure 3.12. In the right part of the figure, a single data acquisition/packet decoding stage is connected with a load balancing filter and three packet/flow export threads which drive various detection engines based on Snort. Based on the constraints discussed above, there are two synchronization levels (dashed sync lines), and a memory pool with three packet batches (left part of the figure) would be reserved for this setup within the data acquisition/packet decoding stage. The packet decoder allocates packets until a batch is completely filled. After that, the reference to the corresponding batch slot is passed to the subsequent packet processing level. The other packet processing levels do the same, and after three passes, the first slot reaches the last packet processing level (lower left part of the figure). Due to the synchronization scheme, the packet decoder in the first processing level can be sure that after the processing of the third slot and its transfer to the next packet processing level the first slot is no longer in use, i.e., it can overwrite the first slot again without the need for synchronization with the many processes/threads of the last level. The threads of the last processing level do not have to synchronize with each other, as they have only read access to the packets.

Thread Activation

Existing parallelization approaches, e.g., the approach in [75] and Suricata, apply threads which *pull* packets/events from the input queues. This thread management scheme has a major drawback regarding the interaction with the operating system kernel. Due to the differences in the individual processing times of threads, there is a high probability that packets in the CPU cache are pushed out during a context switch. This is because threads that are directly connected to each other and are candidates for cache-hits are activated in random order. The modules in this approach use *quasi-synchronous* function calls, i.e., the semantics of the function call is *synchronous* if

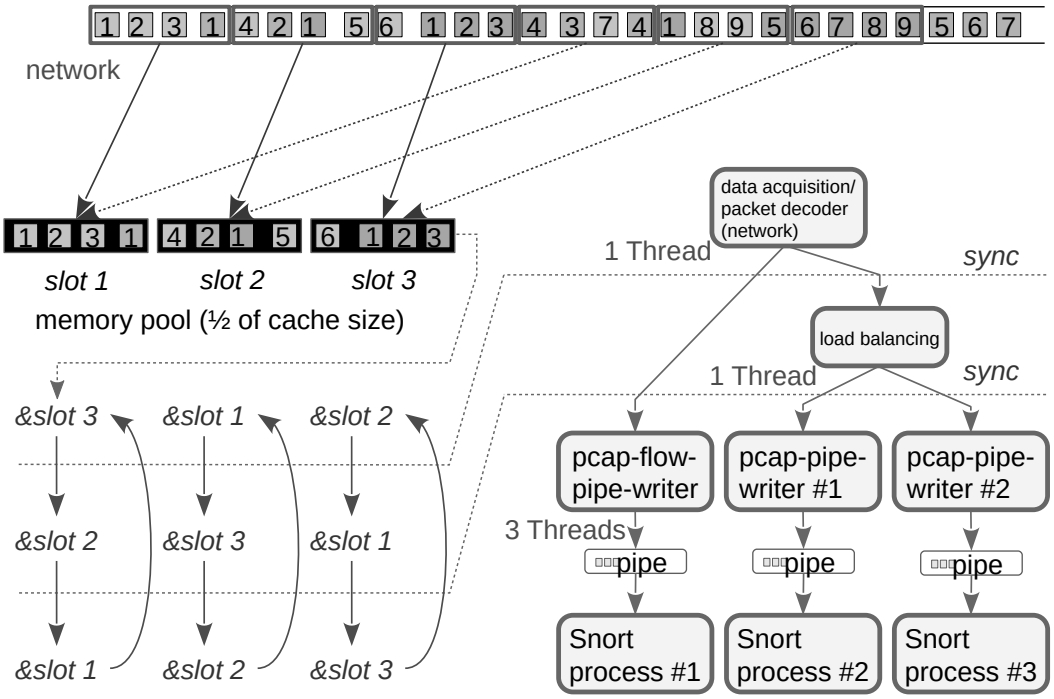

Figure 3.12: Memory management of the architecture

the called module already executes some functionality, and it is *asynchronous* if there are free processing capacities. The resulting call chain (*thread activation*) essentially follows the packet flow through the parallelized modules (cf. Figure 3.11). Thus, the probability increases to keep the packets inside of the CPU cache.

Dynamic Load Balancing

The load balancer of this concept applies a *dynamic* approach. For each incoming network flow (e.g., TCP/UDP flow), the balancer calculates a key k'' for the complete 5-tuple (source and destination IP, protocol number, source and destination port, if applicable) of the flow based on the following hash sequence (which applies a freely selectable hash function):

$$\begin{aligned} k &= hash(seed, transport\_protocol\_number) \\ k' &= hash(k, source\_port \oplus destination\ port) \\ k'' &= hash(k', ip\_source \oplus ip\_destination) \end{aligned} \tag{3.3}$$

This hash sequence allows to *mark* each packet in a batch with the same key for the two communication directions of the corresponding flow and to map the packets to any free module (e.g., detection engine) in the output path of the load balancer. The

latter can apply different strategies to measure the load of its output path, such as counting the number of assigned bytes, flows, the time difference to the previously assigned packet, and blockages of the output path.

Traffic-based Optimization of Flows

There are two types of flows that may have a significant impact on the IDS performance and its detection capability. (1) Network flows with very small transfer units (heavy-hitters, mice flows). For example, a small maximum transfer unit for frames/-packets which is under the control of an attacker, may considerably slow down all IDS threads/instances. (2) Large flows that occur in bursts (elephant flows). They may overload a single IDS thread or instance, while other threads/instances are in an idle state. Usually, there are only few large flows.

The approach can detect the two flow types inside of the load-balancing module by measuring the number of packets and the capacity of each flow. For this purpose, statistics have to be collected for each module in the output path of the balancer, e.g., the longest flow or the flow with the highest number of packets in a time frame. The advantage of this approach is the possibility to measure the impact of these flows on the subsequent analyses (e.g., the detection engines after the load balancer in Figure 3.11). Due to the previously explained parallelism constraints, each module can detect performance bottlenecks by measuring the activities of modules in its output path. If some modules in the output path are still processing a packet batch when a new packet batch has to be processed, then this is a clear signal that the current analysis performance is below the capacity of the input stream.

Reaction to Short-term Bottlenecks

The load-balancing algorithm contains a special condition for reaction on short-term bottlenecks. It measures the activity of modules connected to the output path and rebalances the network traffic if a connected detection engine becomes overloaded. This rebalancing usually requires an analysis of the IDS rules of the related detection module/component to prevent any side effects (e.g., losing state information for application layer analysis). However, according to an analysis of the full Snort signature set by our research group [53], these reconfigurations do not introduce many side effects in practice because the probability of attacks requiring information about the flow state usually decreases with increasing flow size. Attacks that are located further in the flows are usually bound to single packets with no relation to the flow state. Therefore, it is feasible for the IDS analysis to treat the last part of a large flow as stateless. The prototypical implementation of this approach uses Snort as the detection engine. Thus, the load balancer rebalances the network traffic according

to the semantics of the SNORT's TCP engine, which aggregates smaller TCP segments to 64 KB segments. If a module/IDS instance is overloaded and its largest allocated flow exceeds 64 KB, this flow is marked as stateless and is balanced over all output modules/detection engine instances (*flow reconfiguration*).

3.6 Evaluation of the Approach

For estimating the capability of the approach to react to short-term changes of network characteristics, as discussed in Section 3.5, a prototype implementation, as depicted in Figure 3.13, was evaluated on an Intel Xeon machine (E5645@2.4GHz, 6 cores, 12 MB CPU cache, 12 GB RAM). Five coupled SNORT processes are used as detection engines that are attached to pipes transferring PCAP data (*pcap-pipes*). Note that it is not possible to use all CPU cores for the detection engines because the other modules cause some analysis effort (e.g., packet decoding) which should be allocated to a separate CPU core.

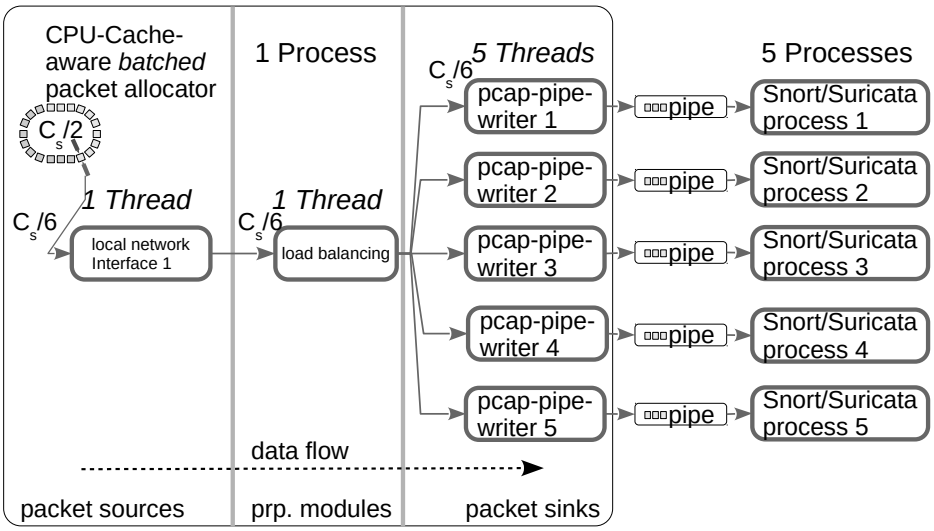

Figure 3.13: Setup of the push-based approach as an external load balancer to SNORT

Performance and Scalability for Small Multi-Core Systems

The parallelization concept was implemented and evaluated in three variants. The first variant (*reconf_lb*) implements all the measures described in this chapter. The second variant (*dynamic_lb*) ignores overloads of analysis units and the third variant (*drop_lb*) discards the largest flow to an overloaded unit. Figure 3.14 shows the prediction of Amdahl for the test system and the average speedup for the three variants of the prototype in a performance comparison to SURICATA. The results show that

the performance of the proposed approach with activated flow reconfiguration (*reconf_lb*) is close to the prediction for the parallelization of SNORT based on Amdahl's formula (*amdahl*). The configuration of the approach without flow reconfiguration (*dynamic_lb*) performs worse because the analyzed datasets represent network flows with bad interleaving, e.g., large flows which occur in bursts and are analyzed by only few detection engines, while the other ones are almost idle. These network flows cannot be balanced evenly over the available IDS instances and the resulting speedup is closer to the prediction for the component-parallel (*comp_par*, cf. Fig. 3.5) variant of SNORT in most cases (e.g., for the analysis of the *nsa_p1*, *nsa_p2* and *defcon* datasets). The third parallelization variant (*drop_lb*) has no advantage compared to the dynamic load balancer. However, the concept significantly outperforms SURICATA for all datasets with the exception of the *defcon* dataset. SURICATA performs slightly better for the *defcon* dataset because it does not analyze the IPv6/TCPv6 packets.

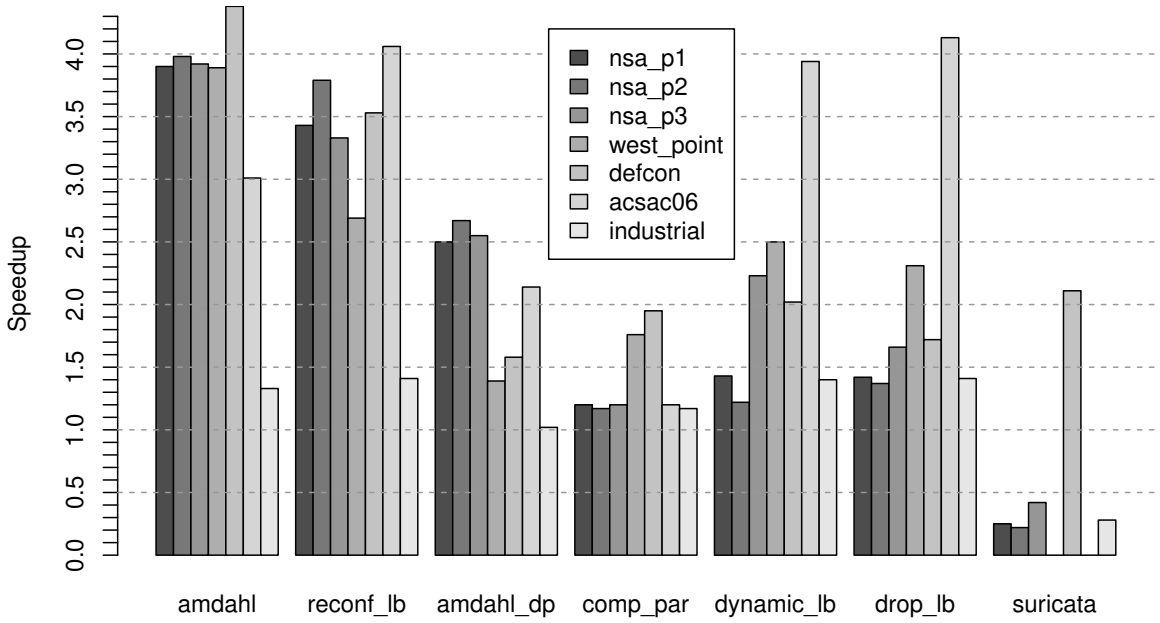

Figure 3.14: Amdahl's prediction in comparison with the push-based implementation with/without flow configuration and SURICATA

Figure 3.15 compares the scalability of the applied measures (load balancing and flow reconfiguration) for a different number of detection engines to the predictions based on Amdahl's formula, with (1) less analysis effort for the serial program part ($r_s = 0.028, r_p = 0.972$), based on the smallest ever measured fraction of the serial program part, (2) more effort for the serial part ($r_s = 0.3, r_p = 0.7$), based on the most often observed distribution for non-attack traffic in literature, and (3) significantly more effort for the serial part ($r_s = 0.7, r_p = 0.3$, cf. dataset *industrial*). As it can be seen from the figure, the prototype scales well and the performance is close to the predictions of Amdahl. The performance gains for the *acsac06* and *nsa_p2* dataset are above the

expected values, whereas the gain for the *west_point* dataset is slightly below, but all curves fall within the expected range of the hitherto recognized distributions between the serial and parallel program part (dashed lines). The anomalies of the industrial data set are discussed in the subsection "Packet Batching and Scheduling Behavior".

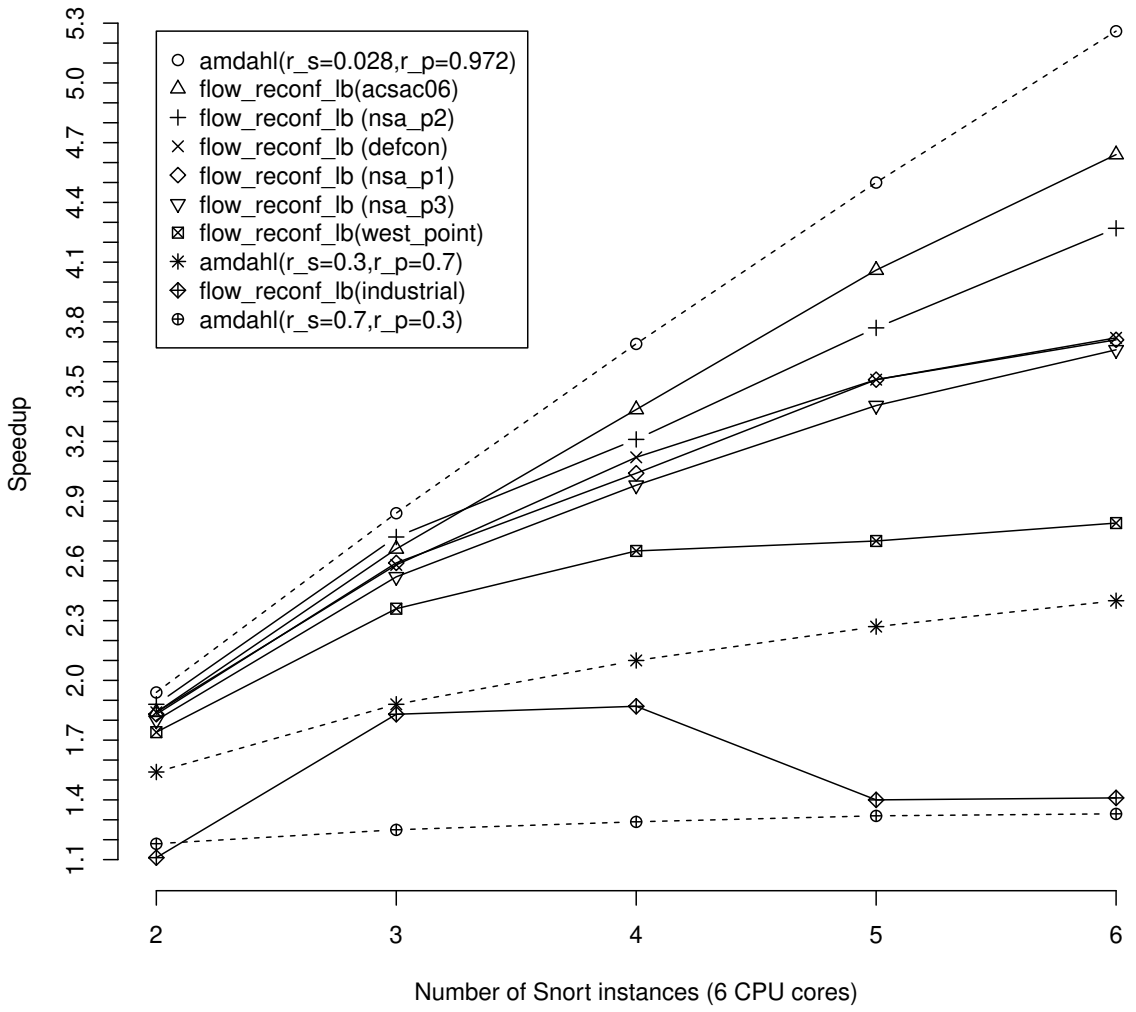

Figure 3.15: Scalability for different numbers of SNORT instances on a 6-core machine

Performance and Scalability for Medium- to Large-Scale Multi-Core Systems

The positive results of parallelization measures for small systems are often not conclusive about the scalability for medium- to large-scale parallel systems. Therefore, the previous experiment was repeated with a much larger data set on a multi-socket system with 20 cores and 40 hyperthreads (2x Xeon E5-2680v2@2.80GHz, 25 MB CPU cache, 128 GB RAM). The data set is 47 GB in size and consists of multiple individual data sets (defcon10, hack.lu, west_point, m57, wireshark, wireshark101, acsac06,

defcon10	hack.lu	west_point	m57	wireshark	wireshark101	acsac06
<————	-darpa_internal————	->	<————	darpa_external————		>
<————————————	maccdc2012————————				-————	>

Table 3.6: Merge/append structure of the combined data set

darpa_internal + darpa_external, and maccdc2012) that have been merged/appended according to the structure of Table 3.6. The *defcon10*[13] data set is 366 MB in size and contains a lot of reconnaissance, e.g., DNS, SSH, and VNC scans, information retrieval via portmap/RPC services, various scans with nessus, SNMP access, brute-force login attempts for FTP and POP3, and many attacks on FTP, as well as attacks on web services, e.g., SQL injection, authentication bypass, and shell access, and furthermore TFTP activity related to malware spreading, and some buffer overflow attempts in various services. This data set is thus densely populated with attacks. The hack.lu[14] data set from a visualization contest is 708 MB in size and contains mostly normal data with the exception of some scans, and a SQL slammer worm propagation. The *m57*[15] record is 4.6 GB in size and is based on a forensic scenario which contains mainly normal data from a test network. The *wireshark*[16] and *wireshark101*[17] data sets are 1.9 GB and 375 MB in size and consist of a mix of normal data and attacks, for example, scan activity, various buffer overflow attempts, spyware code download, and a spread of the SQL slammer worm. *Darpa_internal* and *darpa_external* are data sets that contain normal data and attacks from the well-known (1999) DARPA IDS evaluation data sets[18]. They are 11 GB and 6.4 GB in size and contain IMAP, POP3, SSH and VNC scans, as well as attacks on various FTP, SQL, Telnet and Web services, and various buffer overflow attempts. The maccdc2012[19] data set from the Mid-Atlantic Collegiate Cyber Defense Competition is 16 GB in size and contains a mix of normal business activity and attacks. The attacks comprise scans for various services, e.g., IMAP, POP3, SMB, MS-SQL, Terminal Services, VNC, and VOIP scans, reconnaissance of portmap/RPC services, some FTP attacks, *many* attacks on web services, some buffer overflow attempts, and some trojan activity.

Figure 3.16 shows the scalability of the analysis for the combined data set on the 20-core machine. The number of threads on the x-axis is as follows according to the test setup of Figure 3.13: The load distribution starts with the smallest variant of 6 threads, which includes a thread for input processing, a thread for load balancing, two PCAP-writer threads and two Snort threads. A PCAP-writer thread and the

[13] http://cctf.shmoo.com/
[14] http://2009.hack.lu/index.php/InfoVisContest
[15] http://digitalcorpora.org/corpora/network-packet-dumps/2009-m57-patents/
[16] http://wiresharkbook.com/studyguide_supplements/9781893939943_traces.zip
[17] http://wiresharkbook.com/101_supplements/wireshark101files.zip
[18] http://www.ll.mit.edu/ideval/data/1999data.html
[19] http://www.netresec.com/?page=MACCDC

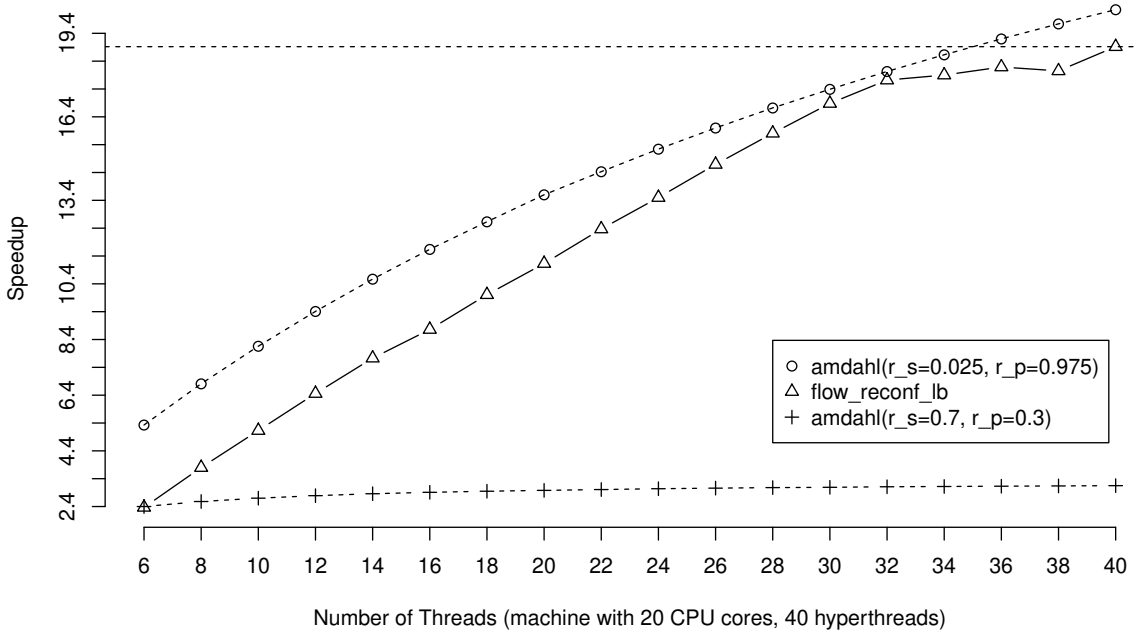

Figure 3.16: Scalability for different number of threads on a 20-core machine

corresponding Snort instance typically jointly run on the same hyper-threaded core. In each step of the figure, this setup is extended by a further PCAP writer thread and another Snort instance. The last step with 40 hyperthreads is equivalent to the maximum setup and includes one input processing thread, one load balancing thread, 19 PCAP-writer threads, and 19 instances of Snort. As in the small-scale experiment, the results are compared with the predictions of Amdahl for the data set's distribution between serial and parallel processing (upper dashed line) and an average distribution for normal data from the literature, which contains no attacks (lower dashed line). As can be seen in the figure, the results follow the curve of the prediction. For a small number of threads, the result is closer to the distribution for data sets without attacks, suggesting that the low degree of freedom for the distribution of the data (e.g., only two Snort instances in the smallest setup) leads to a less optimal load balancing and redistribution of data among all threads. With a larger number of threads that better utilize the system, the shape of the curve is followed more closely.

Packet Batching and Scheduling Behavior

A series of additional measurements evaluated the assumptions regarding the caching and scheduling behavior. For this purpose, the pre-allocated packet buffer was limited to a fixed size and increased by 100 packets in each step. Figure 3.17 depicts the results based on the size of the allocated packet buffers. The results require some additional information about the experiment setup. The test machine (for the small-scale system)

has 12 MB CPU cache and executes a Linux kernel with a CFS (completely fair scheduler). The default thread execution time slice for the applied scheduler is around 20 ms. As it can be seen, the performance initially improves with increasing buffer sizes. This is directly related to the aforementioned time slice of the scheduler. The average packet processing times for most datasets are between 4 and 16μs. If the threads run for at least 20ms, roughly between 1.8 and 7.8 MB have to be buffered (about 1560 bytes for each packet are needed in the implementation). This is where the analysis of the datasets reaches its performance peak/plateau. Based on the same calculation, the performance peak/plateau for the industrial dataset can be expected at 71.6 MB, but the latter is larger than the processor cache. Thus, it is not possible to buffer enough packets for this dataset and anomalies in the scaling behavior can be expected (cf. transition from four to six threads for the industrial set in Fig. 3.15). The upper bound for a reasonable packet buffer size is hard to determine. Depending on the payload of the analyzed datasets, portions of different pattern search automata for Snort have to fit into the CPU cache to achieve a good performance.

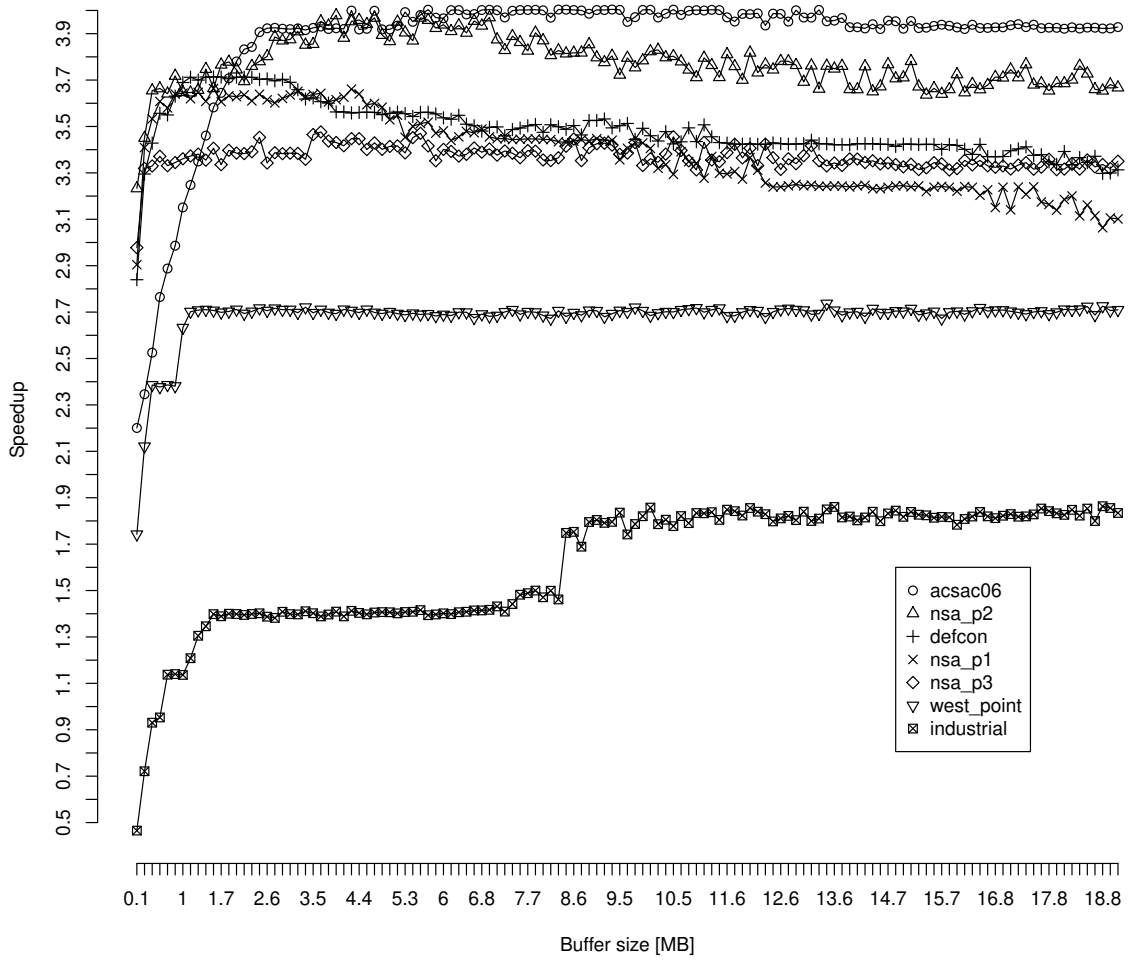

Figure 3.17: Performance for different buffer sizes

Table 3.7 compares the aggregate number of context switches, cache misses, and TLB misses of all SNORT instances and the architecture prototype with the numbers of SURICATA. With the exception of the *defcon* data set, which SURICATA does not analyse at all, the proposed concept causes significantly less context switches (about 6% compared to SURICATA) and less cache misses (about 60% data cache misses and about 40% of the TLB misses).

	cont. sw. [%]	cache misses [%]	dTLB misses [%]
nsa_p1	206,595 [5.7]	335,948,759 [60.9]	261,215,353 [38.9]
nsa_p2	190,664 [5.1]	223,855,945 [64.0]	228,972,563 [35.8]
nsa_p3	183,148 [6.7]	269,267,613 [52.2]	291,503,953 [40.2]
defcon	331,909 [17.4]	599,010,342 [174.1]	352,910,079 [58.6]

Table 3.7: Context switches and cache misses of the prototype [% of SURICATA]

Correctness of the Analysis

For evaluating the correctness of the analyses for the load-balancing approach with flow reconfiguration, the numbers and contents of the detected events (alerts) for all datasets which contain attacks were analyzed. Table 3.8 lists the number of total and unique alerts for each dataset and configuration. The number of total alerts also counts the repeated occurrence of the same intrusion detection signatures, while unique alerts count just one occurrence of each signature. In comparison to the single-threaded SNORT instance, the proposed approach misses some unique alerts (*nsa_p3*, *west_point*, *acsac06*) because the respective signatures have to aggregate different network flows up to a certain threshold, which are now balanced over different detection engine instances. It therefore misses some port scans for SSH, IMAP, and Microsoft's Remote Procedure Call services. However, this behavior is expected from load-balancing approaches and can be remediated by moving the port scan detector in front of the load balancer (cf. flow export filter module in Fig. 3.11). Furthermore, the prototype triggers additional alerts in some cases (total count for *nsa_p1*, *nsa_p2*, *nsa_p3*) because of the configuration of the coupled detection engines. SNORT internally drops flows if the configured memory limit for the flow evaluation is reached (default: 32 MB buffer for flow data). Therefore, the single-threaded variant which has to analyze all parts of the data sets misses attacks that are located later within a dropped flow. The balancing of the flows across multiple SNORT instances just changes the selection of candidates for flow dropping inside of the individual instances and therefore leads to a very small difference in the emitted alerts. It can be concluded from these results, however, that the parallel NIDS correctly analyzes the incoming data despite of flow reconfiguration in overload situations.

	Snort	dynamic_lb	flow_rec_lb	missed_unique_alerts
nsa_p1	8,664 [10]	8,668 [10]	8,668 [10]	none
nsa_p2	2,867 [7]	2,871 [7]	2,869 [7]	none
nsa_p3	9,212 [13]	9,214 [10]	9,214 [10]	SSH/VNC scans
west_pt	603,110 [39]	602,965 [36]	602,976 [36]	TS/IMAP scans
defcon	256,609 [20]	256,580 [20]	256,579 [20]	none
ascsac06	241,420 [41]	240,851 [39]	240,845 [39]	epmap/ms-ds scans

Table 3.8: Load balancing and flow reconfiguration (total alerts [unique alerts])

3.7 Comparison with Related Parallel Approaches

The distinguishing feature of most parallelization approaches is the applied load-balancing scheme and the assumption regarding the synchronization of flow states. There are two classes of attacks which can be distinguished related to their operation sequence and the expense required for their detection. *Multi-step* attacks require a correlation among several flows and thus synchronization of flow states between IDS instances. *Single-step* attacks, in contrast, can be detected with less synchronization efforts. Therefore, the load-balancing strategies can be classified according the following hierarchy: *inter-flow synchronization*, *full intra-flow synchronization*, and *partial intra-flow synchronization*. Inter-flow synchronization, as it is applied in Bro[20], has the highest detection accuracy, but also results in a lower speedup regarding the parallelization. Suricata can be classified into the *full intra-flow synchronization* category due to its static balancing approach that forwards packets of the same flow to a sequential detection engine which analyses them in correct order. The proposed approach with flow-reconfiguration is located between full intra-flow synchronization and partial intra-flow synchronization because it rebalances packets of the same flow in overload situations without synchronization. In the literature, related parallelization efforts usually apply an unidirectional flow concept for stream analysis [126, 128] which is some kind of partial intra-flow synchronization as the two communication directions of a flow are not correlated with each other. Further approaches do not apply any synchronization at all and analyze the packets in a stateless manner [127, 74, 73].

In order to compare the proposed approach with the other solutions a series of additional measurements were performed by replacing the load balancer with (1) two variants of an uniflow balancer (with dynamic and static load balancing), and (2) a stateless balancer. The results of the performance analysis (for the small-scale system) are depicted in Figure 3.18. The prediction by Amdahl and the performance improvement of the proposed approach (*flow_reconf_lb*) are depicted on the left side of the Figure. At the first glance, the stateless (*stateless_lb*) and the dynamic uniflow (*dyn_uni_flow_lb*) balancer significantly outperform the proposed approach.

[20] https://www.bro.org/sphinx/cluster/index.html

Comparing this though with the reported unique alerts in Table 3.9, the stateless approaches miss many attacks, among them buffer overflows, trojan spreading, web-based shell access, and remote procedure calls. Therefore, we can expect that stateless solutions are less able to detect real intrusions. A repetition of the same experiment with the stateless load balancer on the larger system with 20 cores (with the larger data set of table 3.6) detected only 218 of the 748 attack variants (29.8%). The detection accuracy of related work therefore decreases to the same extent as the degree of parallelism increases. Surprisingly, the frequently used static uniflow balancer has no benefits. Presumably, this is because the the subsequent detection engines fall back to the less efficient single packet processing, if the data streams cannot be reconstructed into larger (TCP-)segments. If the TCP stream has been reconstructed, only the communication direction which is relevant for the respective attack is considered (e.g., the much smaller client-request, while the server response often does not contain attack relevant data which is therefore omitted during the analysis in the detection engine).

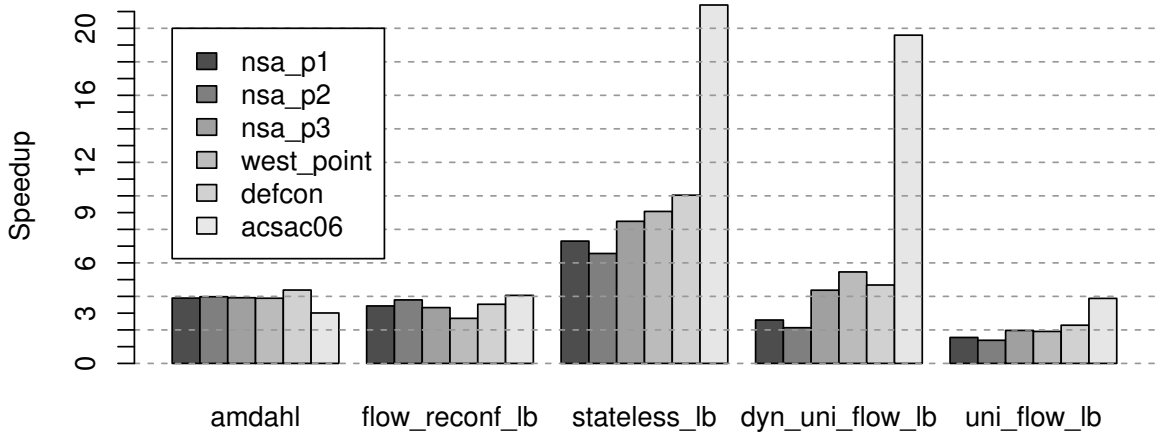

Figure 3.18: Performance increase with disabled flow analysis

	total alerts [unique]		
	stateful lb	stateless lb	missed unique alerts
nsa_p1	8,668 [10]	8,668 [10]	none
nsa_p2	2,869 [7]	2,871 [7]	none
nsa_p3	9,214 [10]	9,215 [9]	none/1 (non-deterministic between runs)
west_pt	602,976 [36]	602,871 [30]	buffer overflows, network bind of cmd.exe, access file download, script file upload, Terminal services/IMAP/WEBDAV scans
defcon	256,579 [20]	257,086 [20]	none
acsac06	240,845 [39]	157,372 [27]	scans of various MS network services, overflows of various MS network services sasser/korgo/ms-blast/lovegate trojans

Table 3.9: Missed alerts – stateless compared to stateful load balancing

3.8 Conclusions

This chapter has investigated different approaches to speedup NIDS analysis capabilities. Parallelization is one of the most important approaches to improve the analysis performance, but existing solutions do not provide the expected performance gains. There are various reasons for this, such as time-consuming memory access patterns, excessive interaction with the operating system kernel, implicit synchronisation of threads by means of the memory allocation strategy, and bad cache-sharing behavior among multiple threads. It was suggested to reconsider the architecture of current network intrusion detection systems and a novel concept has been proposed that allows one to react to performance bottlenecks in a very short time interval in which current intrusion detection systems fail. The proposed approach applies a CPU-cache-aware packet allocation strategy with a thread activation scheme based on quasi-synchronous function calls that essentially follow the packet flow. Furthermore, packets are processed in batches instead of invoking one thread for every packet to optimize interactions with the operating system kernel and the cache locality of the applied methods. The application of the dynamic load balancing concept to several SNORT detection engines combined with a flow reconfiguration in case of performance bottlenecks has shown significant performance gains which are close to the theoretical maximum as predicted by Amdahl's formula without loss of detection accuracy.

As predictions based on Amdahl's formula already have indicated, further performance improvements on a single computer system cannot be expected. The only way to further improve the performance is a distributed system with additional capabilities to discard network data on individual components. For doing so, a couple of new research challenges arise discussed next.

Packet dropping. The signature database of Snort contains indications on the expected location of attack data in a data stream. This information can be used to discard parts of the data streams before they are forwarded to the analysis. The attack data are, for example, often located at the beginning of client requests. Limmer et al. have used this characteristic in the *Dialog-based Payload Aggregation* (DPA) [138] to forward only the first n bytes whenever a TCP connection starts or the direction of the data transfer changes. This approach seeks to discard as much data as possible and to maintain a defined detection accuracy. The number of packets to be dropped could, however, also continuously be calculated via a function based on a history of congested analysis units. The function would have – similar to the TCP flow control – to determine how many packets can be forwarded from each batch.

Protocol/Service dropping. The NIDS signature base includes well-known attacks for a defined set of communication protocols, applications, and operating systems. So far, only fragments of the communication protocols and a few application-identifying attributes have been described by attack signatures. The result is

that the NIDSs sometimes analyze data for which there are no signatures. If it would be possible to identify individual connections after an initial protocol analysis with regard to the communicating applications/operating systems, individual connections could be dropped if no signature exists for them. This can reduce the amount of traffic to be analyzed and increase the accuracy of the analysis (reduction of false alarms due to incorrectly parsed protocol fragments).

Fast hardware-based flow dropping. Amdahl's formula predicts that the sequential part of the analysis is the ultimately limiting factor for parallelization. On a general-purpose CPU, nothing can be changed regarding the speed of this part in relation to the parallel parts. It is, however, possible to outsource the (sequential) decoding of network packets to special hardware which can perform the necessary calculations/analyses faster.

Analysis of global data flows. The analysis of global data flows which cross several network switches/routers can be distributed across multiple monitoring points in the local area network, i.e., the analysis of individual and logically related data flows can be assigned to less busy analysis units which observe the same data flows. For this purpose, strategies must be developed to determine the optimal initial location of analysis for each flow and a distributed algorithm is needed which divides the load among the monitoring instances. In addition to the analysis acceleration, the global analysis can be used to uncover complex relationships in the analysis results. Attacks often consist of multiple attack phases that may trigger varying alarms on different analysis units in the network. It would be interesting to see whether clustering algorithms can relate the attack activities in different data flows to each other to uncover their relationship with respect to a common goal.

Parts of the above described functionality as well as other innovative functions can also be realized through a combination of NIDS and SDN technologies. The SDN controller from Chapter 2 could be extended with another security service which controls the monitoring of packets/flows to a NIDS. First ideas on this subject were evaluated by Amann et al. [139] using the Bro Network Security Monitor.

NIDS load reduction. The basic idea is that the NIDS determines which packets are forwarded to the NIDS sensor. The NIDS signatures typically do not cover all data streams that can occur within a network. In a first step, the above-mentioned protocol/service analysis could determine, for example, which data streams are not covered by the signature set. If a prolonged data stream occurs (e.g., a file download), the SDN controller could be instructed by the NIDS not to mirror the corresponding packets anymore. Amann et al. have determined that using this technique up to 53% of TCP data could be kept away from the NIDS in the surveyed network [139].

Dynamic NIDS-based firewalling. The NIDS analysis results can also be used to mitigate the impact of attacks. Recognized attack sources could be isolated from the network, for example, by means of the SDN controller. If this reaction is too strong a SDN-based rate limiting could gain time for a more detailed analysis of potential attack sources, e.g., hosts that perform port scans. In addition, recognized attack sources could be diverted to a special honeypot to analyze the next steps of the attackers.

4 Firewalls and NIDS for Web Applications

Modern web applications and online services intensively use Web 2.0 technologies which require running active content in the browser of the users. Unfortunately, these technologies are often misused for attacks. Switching off web applications to better protect the systems is not an acceptable way out because it is usually not desired by the users. Current perimeter firewalls allow it only to a limited extent to filter out unwanted traffic. If a blocking of web technologies is not desired, firewalls should be able to detect active contents in data streams, to extract them, and to examine them for malicious code. This chapter proposes a novel architecture for a client-side/server-side web firewall with an accompanying analysis library that extends the firewall analysis by capabilities to normalize web traffic at the application layer, to detect firewall evasions, and to classify web applications using machine learning methods with the objective to identify the web applications and to selectively pass them according to the given firewall policy. The approach aims at mainly supporting small and medium-sized enterprises which usually use and run a restricted number of web applications. The firewall is supposed to be located in the perimeter of the network to protect the clients (browsers) or in a demilitarizied zone (DMZ) to protect smaller web application installations and the accompanying analysis library can also be used independently on the server side in a web application, or on the client side in the browser. The full firewall allows only access to a defined set of web applications for clients within the network and removes active code from web pages otherwise. In principle, some of these methods can be implemented using URL filtering. Experience though has shown that such filtering modes are very unreliable because it is often possible to write any domain into the HTTP *host* field, while the web server delivery of the page is not denied.

Since there are well-established and easy-to-implement solutions to mitigate *cross-site request forgery* (CSRF) attacks, such as CSRF tokens, the focus is more on *cross-site scripting* (XSS) attacks. Although browsers have built-in security mechanisms to detect reflected XSS attacks, in which a part of the client input to the server is contained in the response, they are helpless against stored XSS attacks, which deliver malicious code from the server not included in the request. The concepts presented here differ from previous research in the following points:

- The analysis is performed in the *perimeter firewall* when accessing a web page. Modifications or extensions of the browser, server or both sides are not required.

- Communication fragments are analyzed independent of the context in the browser and thus independently from a specific browser implementation to protect all browsers.

- In contrast to other works that require accurate knowledge of the structure of an attack (e.g., [140, 89, 90, 94, 141, 142]) and thus probably only recognize specific attacks, the presented approach separates benign code from a corpus of other benign code to detect anomalies and therefore possibly unknown attacks.

4.1 Cross-Site-Scripting Attacks

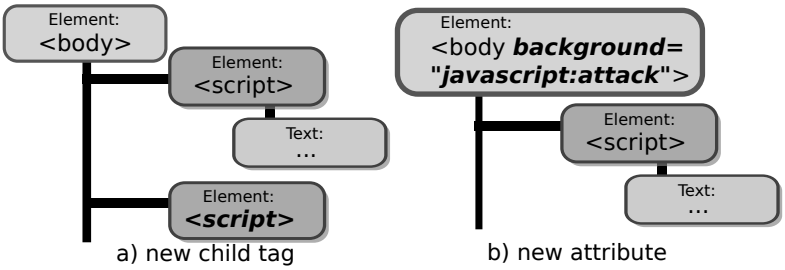

Figure 4.1: Structural changes of stored XSS

Cross-site scripting (XSS) allows attackers to inject client-side code into web applications executed by other users. There are several types of cross-site scripting attacks. The most common ones are stored and reflected XSS. *Stored XSS* assumes that the attacker can directly modify the web application, e.g., by including code in a forum entry. This enables changes, as they are shown in Figure 4.1 (bold italic changes come from the attacker). Depending on server-side defenses, an attacker typically can inject new tags with active code (left side) or enhance attributes of tags to incorporate new code (right side).

Reflected XSS assumes that the attacker has no direct access to the web application. Instead, code is sent piggybacked through user input to the web application's server and returned after server-side processing as client-side code to the unsuspecting user. Typical variants of reflected XSS attacks are *node splitting, attribute splitting*, and *tag splitting* (cf. Fig. 4.2). They are used to split the document structure of the server-side output in unprotected areas and to inject own code instead of the originally planned user input into the respective sections.

A third type of cross-site scripting that has emerged with the Web 2.0 is *DOM-based XSS*. Here, the injection is in an URL fragment that is processed at the browser side and never sent to the server. Since the focus is, however, on perimeter firewalls and NIDS, this attack type is out of scope of this work. Section 4.2 discusses other client-side measures to prevent such kind of vulnerabilities.

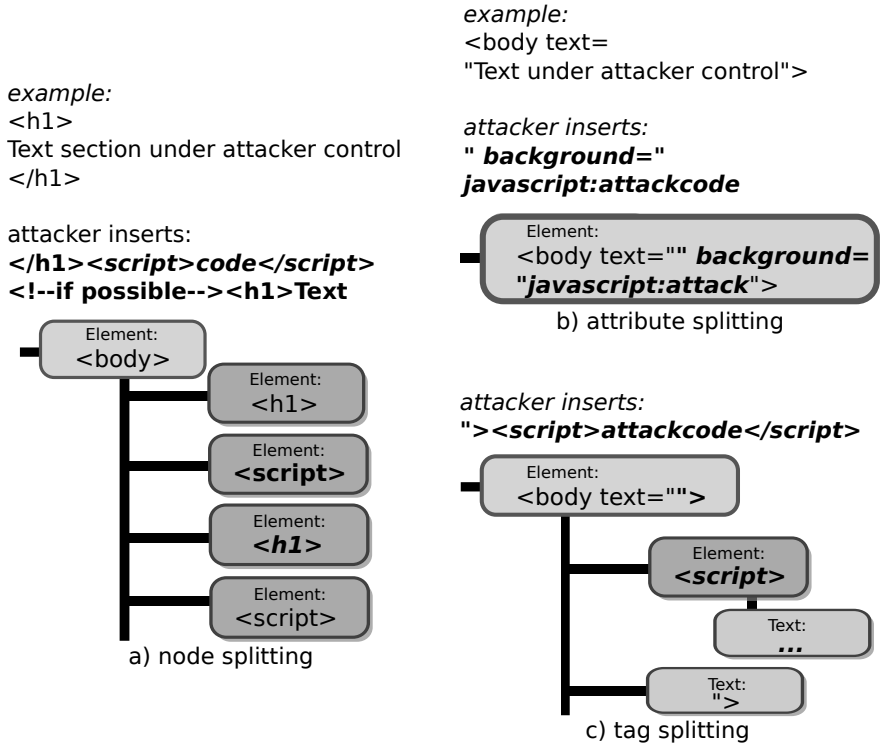

Figure 4.2: Structural changes of reflected XSS

4.2 Current Approaches to Improve Web Security

The research on the protection of web applications includes client-side, server-side, and hybrid approaches. In [89], for instance, Hallaraker et al. propose a client-side audit framework for JavaScript in Mozilla which can be used to construct signature-based or anomaly-based intrusion detection systems (IDSs). They also propose first basic signatures to detect information extraction (cookies) and aggressive scripts (e.g., prevention of window closing). This work is continued in [90] by Vogt et al. who add dynamic data tainting to the client side to track the usage of sensitive data sources (e.g., the aforementioned cookies) in control statements to reveal malicious indirect data leakage. The first real anomaly-based detection approach is the browser extension JaSPIn [92] that creates a profile of the application usage of JavaScript and enforces it later. Unfortunately, these profiles are bonded to both the behavior of the website, and to the browse behavior of the user. If the web page changes or the user changes his/her behavior, the profile must be adjusted again. The "Lightweight Self-protecting JavaScript" approach by Phung et al. [143] allows the web developer to write server-side policies, that are enforced by embedding a client-side JavaScript library which overrides the native browser functions. This dynamic approach, however, is associated with an average slow-down of the code execution by a factor of 6. Document structure integrity [144] is an approach in which the web application developer or a server-side

taint tracker (indirectly) marks all document areas which are affected by user input. These areas are represented as text nodes when parsing on the browser side, thus preventing the execution of active code. A particular problem with this approach is the identification of user inputs. In *DOM-based XSS*, these inputs are made on the client side and do not reach the server, resulting in that they are not covered by the server-side taint tracker. ALHAMBRA [145] supports both document structure enforcement (limited to elements with a *src* attribute) and JavaScript analysis based on client-side taint tracking to prevent DOM-based XSS. The idea of client-side taint tracking was later refined and elaborated in detail by Stock et al. [91]. The results indicate that this is a promising approach to eliminate DOM-based XSS. The CSFIRE [93] browser extension by Ryck et al. removes authentication information from most cross-site requests to prevent *CSRF*. It only permits cross-site requests when they are used together with payment or single-sign-on solutions based on an algorithm that detects the respective traffic delegation patterns. The browser-based approach ZOZZLE [94] analyzes the JavaScript abstract syntax tree with a Bayesian classifier to detect malware. The feature selection bases on the χ^2 algorithm that examines the correlation of several features with the underlying malware samples. However, it can detect only one type of attack (heap spraying) and if the malware does not contain the stochastically reduced feature set the tool cannot detect attacks any more. The ICESHIELD approach of Heiderich et al. [141] uses a linear decision function that differentiates malicious code from normal code based on heuristics for several attack types that apply code obfuscation. The necessary detection technology is embedded directly as JavaScript into the web page and the detection functions are protected by means of special JavaScript attributes against manipulation by the attack code. The reported false-positive rate for this approach of 2.17% for 61,504 pages seems to be quite high. Another approach by Rui Wang et al. [142] also detects XSS based on code obfuscation. In this case, the obfuscation measures are sorted into groups (for example, string operations) and a decision tree based algorithm determines based on the aggregated (grouped) features, whether a HTML or JavaScript document is malicious. Both approaches are likely to fail if the attacker does not use obfuscation measures.

Server-side web security approaches usually require modifications of the web applications, sometimes with mandatory support from the client-side. Approaches, such as BEEP [87], BLUEPRINT [88], CONSCRIPT [146], and the *Content Security Policy* let the developer specify a security policy that has to be enforced at the client. These policies allow one to distinguish between application-specific and external data, and may even restrict executions inside the JavaScript interpreter. Other approaches, e.g., NONCESPACES [84] and S2XS2 [86] provide a framework in which the developer has to explicitly distinguish between trusted application data and untrusted external data. He/she has further to specify how untrusted data can propagate into a dynamically generated page. Based on these information, the server then can restrict the propagation of unexpected external data. Another approach, SWAP [85], relies

on a server-side proxy, server-side (modified) browser, and an initial web application profiling at installation time to distinguish between legitimate and malicious scripts.

Proposals for intermediate systems, e.g., firewalls, are rare, but there are some solutions which could be used with some modifications on intermediate devices. A first approach of Ismail et al. [140] applies signature recognition by augmenting typical XSS fragments in a page request (e.g., < <) with random numbers and searching for them in the result page. NOXES [147] works as a client-side proxy that tries to protect against data leakage through detecting and white-listing valid cross-site requests by analyzing the requested web page. It has, however, a high false-positive rate and often requires user interactions. The server-side proxy XSSDS [148] tries to learn the application-specific JavaScript of the original application to accept later only known JavaScript. Additionally, it facilitates to define filters against *Reflected XSS* attacks similar to the currently used inside web browsers. The method can detect *Stored XSS* to a limited extent by comparing stored tokenized code fragments with the actual code, but it is not resilient against changes of code blocks. A similar approach of comparing the received page with the expected one inside a server-side proxy is applied in XSS-GUARD [149]. Instead of completely learning the application-specific script, a shadow page based on benign input is generated using an adapted application. The shadow page is then compared with actual pages. The problem with this approach is that the comparisons are based on a modified (Firefox) browser engine GECKO. Therefore, it is not clear whether it can detect attacks which are specifically destined for other browsers.

4.3 Non-Applicability of Classical NIDS Methods

Today's web security is essentially based on server-side checks of inputs to web documents. However, many server-side checks fail because of subtle differences in the input format, in the intermediate document representation (syntax), or the final presentation in a browser. The languages and data structures used in the Web 2.0 are based on a number of historical standards and proprietary extensions that have, however, left large room for interpretation regarding the presentation, functionality, and error handling during the development stage. The original Standard Generalized Markup Language (SGML[1]) includes a method to mark up the description of the document structure (Document Type Description - DTD) and to output the document on different media (Document Style Semantics and Specification Language - DSSSL), e.g., screens with different resolutions or printers. The markup language defines a method for structuring documents and provides rich formattings of text fields. Elements are, for instance:

[1] `http://www.w3.org/MarkUp/SGML/`

- opening tags:
  `<ELEMENT NAME>`
  → e.g., (HTML) `<HEAD>`

- closing tags:
  `</ELEMENT NAME>`
  → e.g., (HTML) `</HEAD>`

- item attributes:
  `<ELEMENT NAME attr1 = "value1" attr2 = value2 attr3>`
  → e.g., (HTML) `<INPUT type = "password" class = pw autofocus>`

- a proposed structure for nesting areas:
  `<ITEM1><ITEM2></ITEM2></ITEM1>`
  → e.g., (HTML) `<BODY><SCRIPT></SCRIPT></BODY>`

Both the Hyptertext Markup Language (HTML) and the Extensible Markup Language (XML) were derived from the text markup language SGML. The Document Type Description Language was adopted for these languages as well. A relationship between SGML and HTML was defined for early variants of HTML, but not for XML. HTML and XML have their own formatting and output languages (Cascading Style Sheets – CSS or Extensible Stylesheet Language – XSL, and XSL Transformation). In this case, the Cascading Style Sheets may be used within XML. From XML, XHTML was derived later. As a result, attackers can use four language generations for the text markup (SGML → HTML → XML → XHTML), including the extension of the document structure (DTD) and the output formatting (CSS, XSL, XSLT). The dependencies among these languages are unclear to some extent and give room for different interpretations. Thus, an attacker can create mixed documents which are based on different language generations and bypass analysis systems which can usually analyze only one language (generation) at a time.

This is exemplified in Figure 4.3 with the help of two modified examples taken from the HTML5 Security Cheat Sheet[2]. The document declares itself in the first example as an extension of the XHTML standard and defines an implicit attribute for the *img* tag whose value is based on the JavaScript language (*onerror* is invoked in case of errors, e.g., for the specified incorrect image *src* attribute). The calling browser then executes the JavaScript code for each custom image tag. The second example contains an extension of the XML standard (*x:script* → XML namespace) that in turn can be used for the evasion of protection measures. Analysis systems that detect active code using simple regular expressions (e.g., *<script.*</script*) would fail to detect the example.

[2] `http://html5sec.org/`

```
<!-- Example 1 -->
<!DOCTYPE x
[
<!ATTLIST img
     xmlns CDATA "http://www.w3.org/1999/xhtml"
     src CDATA "xx:x"
     onerror CDATA "alert(1)">
]>
<img />

<!-- Example 2 -->
<x:script xmlns:x="http://www.w3.org/1999/xhtml">
     alert(2);
</x:script>
```

Figure 4.3: Analysis evasion using XHTML

The examples in Figure 4.3 require a correct XML formatting. However, attackers often evade server-side checks using incorrect HTML formats. This is exemplified in Figure 4.4, which is also partly based on the HTML5 Security Cheat Sheet. In the first example, self-closing *script* tags open sections with active code that are closed with a broken tag in the second case. The second example uses a text node and a JavaScript comment to hide the surrounding *script* tags from simple document parsers. The third example circumvents regular expressions which try to rule out parameter values with evenly balanced quotation marks. The last example skips the closing angle bracket to circumvent naive regular expressions and simple document parsers at the same time. In addition, attackers can circumvent protection measures using manipulated or infrequently used character encodings for text-based web documents, such as:

- 7-bit encodings, e.g., in HTML with the construct ¼script¾alert(1);¼/script¾, in which the eighth bit in the encoding *x-mac-farsi* is deleted. The result is interpreted as a *<script>alert(1);</script>*.
- Half-width Unicode, e.g., in HTML and CSS with the construct *<style> * {x: U+FF45 U+FF58 U+FF50 U+FF52 U+FF45 U+FF53 U+FF53 U+FF49 U+FF4F U+FF4E (write (1))} </style>*, in which some letters represent a multi-byte encoding (decimal > 255) forming the term *expression*, but which are mapped to the ASCII set (0-127 decimal) by some parsers.
- Invalid/non-printable characters, e.g., in HTML, with the construct *XX*, in which the lack of error handling of many browsers is exploited, which just swallows non-printable or invalid encoded Unicode characters, even if the encoded area is security-critical.

```
<!-- Example 1 -->
<script/>alert(3);</script>
<script/>alert(4);</script/>

<!-- Example 2 -->
<<SCRIPT>alert(5);//<</SCRIPT>

<!-- Example 3 -->
<IMG """><SCRIPT>alert(6)</SCRIPT>">

<!-- Example 4 -->
<IMG SRC=x onerror="javascript:alert(7)"
```

Figure 4.4: Analysis evasion using malformed HTML

Considering these evasion possibilities, it seems to be obvious to create rule sets for existing NIDSs or content-based firewalls that detect these attempts. However, a pure signature-based analysis based on classical NIDS methods often does not work as expected. Consider, for instance, the following example from the NIDS Snort in Fig. 4.5. The depicted signature detects an explicitly via the Help Center (hcp://) loaded document with an embedded struct < *script*... that has both an ASCII (\x3c) and an Unicode-based (\x253c) encoding of the opening angle bracket, various ASCII space characters (\s), and an Unicode (\x2520) space. Under real attack conditions, this signature can only identify two variants of the attack, while many parser implementations additionally accept all non-printable ASCII characters like \x0, \x9, \xA, \xB, \xC, \xD (NUL, horizontal tab, new line, vertical tab, form feed, carriage ret), the mentioned slash (/) from Fig. 4.4, non-printable Unicode characters \x00A0, \x2000–\x200A, \x2028, \x2029, \x202F, \x205F, \x3000 (spaces of different widths, line separator, paragraph seperator, narrow no-break-space, medium mathematical space, ideographic space), and invalid Unicode characters, such as \x5760 and \x6158 as a separator or whitespace. To add these additional variants would not be effective because this explosively increases the signature base within a very short time. The encoding problem is getting worse when it comes to the detection of problematic JavaScript fragments, e.g., when searching for constructs, such as *eval*. Figure 4.6 illustrates an encoding for the code *eval(alert(1))* with a JavaScript program that does not use any alphanumeric characters[3].

The second problem with traditional NIDSs is that they are designed *to prove that code* instead of normal application data *is transferred over the network*. The problem with web applications is that *they are designed to transmit code via the network* and now we have to *prove whether this code is benign or malicious*. An example of malicious

[3]encoded using jjencode: `http://utf-8.jp/public/jjencode.html`

```
alert tcp $EXTERNAL_NET $HTTP_PORTS -> $HOME_NET any
(flow:to_client,established; file_data;
 content:"hcp|3A 2F 2F|";nocase;
 content:"script";distance:0;nocase;
 content:"defer";distance:0;nocase;
 pcre:"/hcp\x3a\x2f\x2f[^\n]*(\x3c|\x253c)script
       (\s|\x2520|\x2f)+defer/iO";
 sid:16665;)
```

Figure 4.5: Snort IDS signature for an attack on the Windows Help Center

```
$ = ~[];
$ = {___:++$, $$$$:(![] + "")[$], __$:++$, $_$_:(![] + "")[$],
_$_:++$, $_$$:({} + "")[$], $$_$:($[$] + "")[$], _$$:++$,
$$$_:(!"" + "")[$], $__:++$, $_$:++$, $$__:({} + "")[$],
$$_:++$, $$$:++$, $___:++$, $__$:++$};
$.$_ = ($.$_=$ + "")[$.$_$] + ($._$=$.$_[$.__$])
+ ($.$$=($.$ + "")[$.__$]) + ((!$) + "")[$._$$]
+ ($.__=$.$_[$.$$_]) + ($.$=(!"" + "")[$.__$])
+ ($._=(!"" + "")[$._$_]) + $.$_[$.$_$] + $.__ + $._$ + $.$;
$.$$ = $.$ + (!"" + "")[$._$$] + $.__ + $._ + $.$ + $.$$;
$.$ = ($.___)[$.$_][$.$_];
$.$($.$($.$$ + "\"" + $.$$$_ + "\\" + $.__$ + $.$$_ + $.$$_
+ $.$_$_ + (![] + "")[$._$_] + "(" + $.$_$_ + (![]
+ "")[$._$_] + $.$$$_ + "\\" + $.__$ + $.$$_ + $._$_
+ $.__ + "(" + $.__$ + "))" + "\"")())();
```

Figure 4.6: JavaScript program “eval(alert(1))” encoded exclusively with symbols

code is a code-based denial of service attack on a browser. In order to detect such an attack one would have to be prove that the transmitted code fragment finishes its execution. This is the classical halting problem for which Turing already proved [150] that it is not solvable. Another approach – to write context-sensitive signatures based on the context-sensitive parsers to identify *malicious embeddings* of code – was also tried during the research for this work, but almost every web page contains one or two harmless syntax errors that trigger the corresponding rules. Due to the discussed problems, an approach is sought which can *prove the integrity of the transmitted code* instead of its benignity or maliciousness. Only a domain-specific model that uncovers subtle differences between the transmitted code and the known coding style of the web application by means of machine learning methods will be capable to solve this problem.

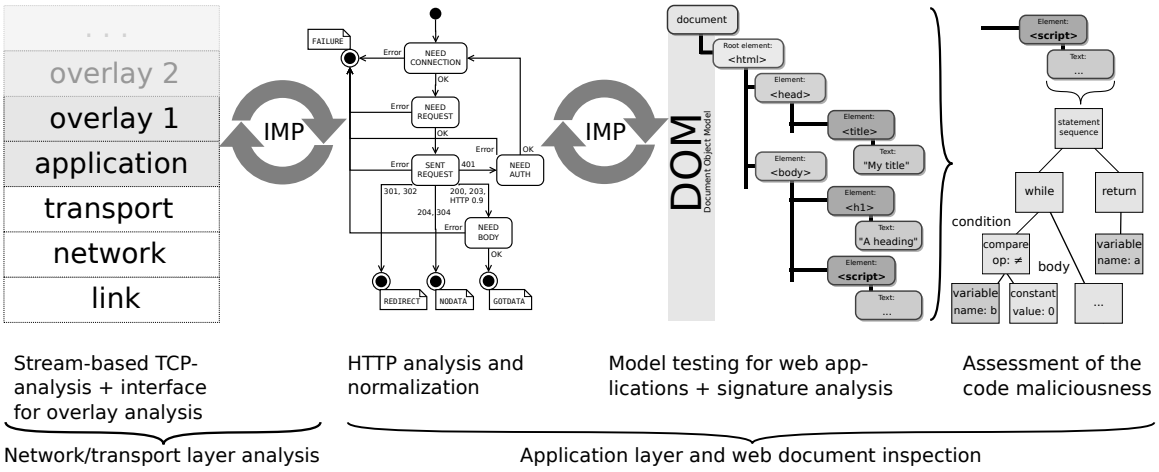

Figure 4.7: Firewall architecture for web applications

4.4 Web Analysis for Application Firewalls

The broad range of analysis evasion possibilities requires novel approaches to protect web applications. The basic idea behind the approach proposed in this section is to combine preemptive security methods, e.g., firewalls, with reactive ones, e.g., intrusion detection analysis methods. For this purpose, an intrusion detection analysis unit should be linked to the firewall that is able to parse the application protocols and the highly hierarchical Web 2.0 document structures and languages, so-called (application protocol) *overlays*, to detect and block malicious or unknown active contents (see Figure 4.7). For the synchronization between the the firewall and the analysis unit, a specific *inspection and modification protoco*l (IMP) is used.

The proposed analysis system includes three operating modes – a training mode, a validation mode, and the active/effective operation mode. In training mode, the structure of the web documents (HTML, JS) for each domain is transferred by means of parsers into feature vectors for a machine learning system that are stored in a database. If enough feature vectors are available, e.g., after a day of operation, multiple application models for each domain are generated offline by means of machine learning with different parameters. Thereafter, the system proceeds in the validation mode that tests the different models for each domain online and fixes the parameters for the best models. Finally the system goes into the active/effective operation mode which validates new or unknown active content of a domain against the best domain model. The domain model has to prove whether the new code or a new document corresponds to the "style" of the respective web application, e.g., Facebook.

The response to deviations from the style of a web application depends on the intended deployment scenario. The main scenario of this approach is a perimeter firewall that

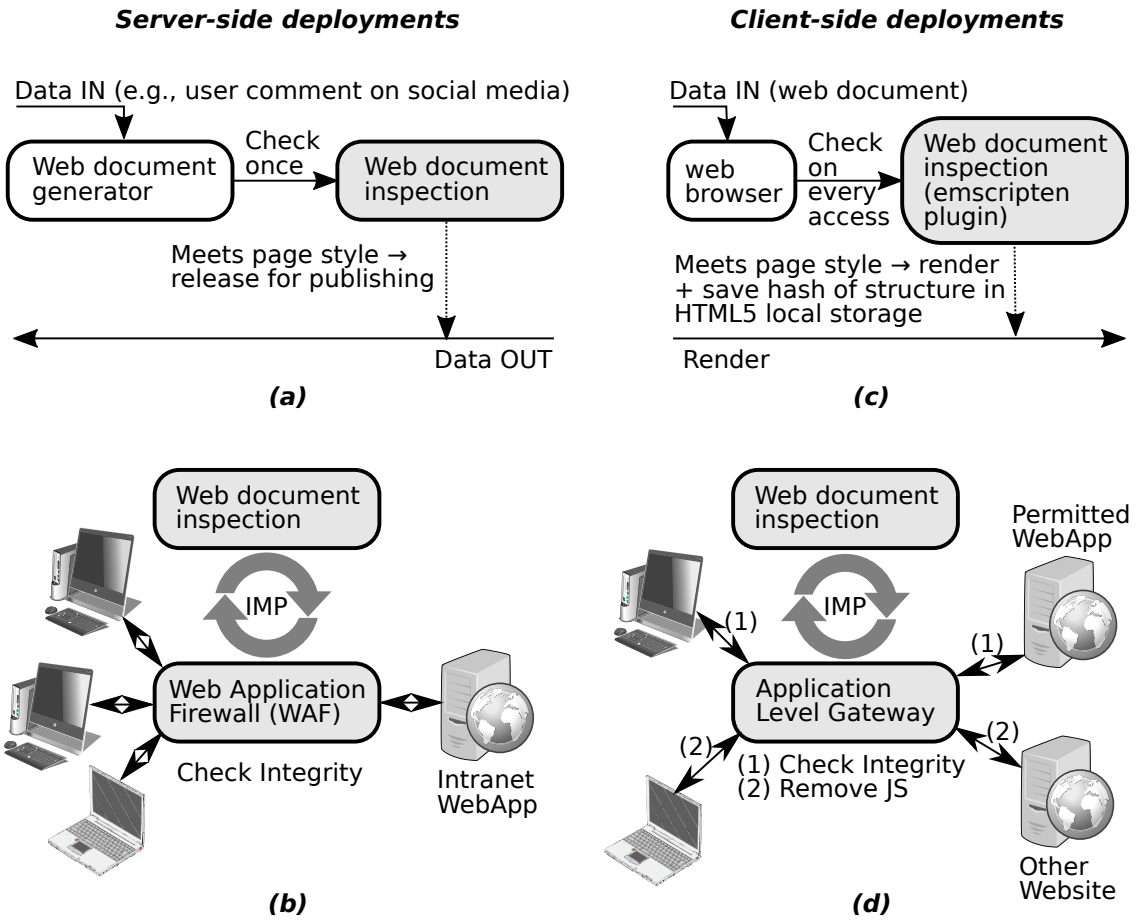

Figure 4.8: Deployment scenarios for the analysis units

covers all components from the analysis of the application layer (HTTP) to the analysis of documents. However, the increasing use of encryption could thwart this scenario in the future. Therefore in the second scenario, only the web document analysis library is used to support client- or server-side integrity checks for web pages which are exposed to user input. Figure 4.8 represents two server-side and two client-side use cases based on these scenarios. In the first server-side scenario (a) a high-traffic page is assumed to which only the web document inspection (without application layer protocol inspection) is applied to check user-generated content. If the page that is generated from the user input matches the style of the overall domain, it is released for publication. Otherwise, it is blocked. The second server-side scenario (b) assumes a protected small web application (e.g., an intranet application), which is checked for integrity by a web application firewall that implements the whole architecture (application layer and web document inspection). Typical web application firewalls check inputs to the application, add and verify CSRF tokens, harden session cookies, and detect common attack patterns. Usually they require to be tightly integrated with the web application (e.g., they require knowledge of the input types to forms of a specific web application)

and can thus provide only an insufficient generic protection. When combined with the proposed learning-based approach the transmission of the application protocol or document can be stopped if deviations from the style of the web application are detected. The first client-side scenario (c) applies the document inspection analogously to the first server-side scenario to test individual web applications for integrity. In this case, the models are exclusively generated for the most relevant web applications, e.g., applications containing pages with password fields. The models, the tree hashes of the web page structures, and the analysis results of known pages of the protected web applications should be kept in the local HTML5 storage. The last client-side scenario (d) assumes a high-security network in which active code is generally blocked or removed. Only a small amount of web applications is enabled to pass through the firewall. The application layer gateway constructs a model of each application and removes active code from all documents that do not match the style of the respective application. This scenario is identical to the architecture depicted in Figure 4.7. More in detail, the following steps are performed in the full firewall architecture.

Signature Analysis and Normalization for HTTP

In the first step, transport layer data are passed to the HTTP analysis and normalization unit that is mainly dedicated to rewrite or remove headers to prevent attacks on the application (HTTP) layer. A proprietary protocol called IMP (inspection and modification protocol) was developed for this transfer. IMP transmits individual data blocks to the analysis component as soon as they have been recorded by the firewall, i.e., it does not wait on the completion of the entire data stream (in contrast to ICAP [151]). In this way, abnormalities can be detected early and the web data stream can be blocked. The synchronization takes place via the position of the data in the original data stream.

Normalization means that the protocol requests and responses are rewritten to match the correct protocol behavior. The analysis unit roughly follows the internal HTTP client state machine, as depicted in Fig. 4.9. In protocol states in which no data is expected (e.g., transition to NODATA for HTTP response codes 204, 304), headers, such as content-length and transfer-encoding are removed. For other states, e.g., the transition from SENT REQUEST to GOTDATA via NEED BODY, the information of these headers is replaced by the observed length and encoding. Duplicate headers are replaced through only one instance and their content is normalized according to the observed protocol behavior, e.g., additional data after the transition to NODATA is removed from the stream.

In order to block malicious cross-site requests (CSRF) two additional modules are connected to the firewall. The first module[4] removes session credentials (cookie,

[4] `https://github.com/noxxi/p5-app-http_proxy_imp/`

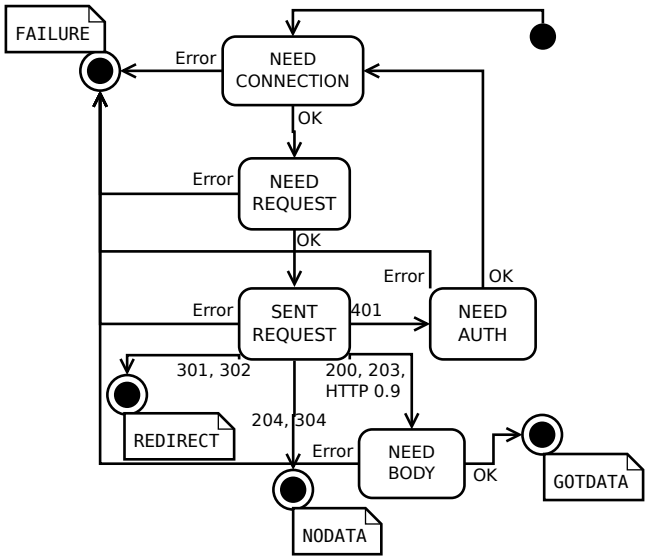

Figure 4.9: HTTP client-side state machine (based on `https://www.w3.org/People/Frystyk/thesis/HTTP.gif`)

cookie2, and authorization header) from the request if the origin of the request is not known/trusted. The origin is determined by the *Origin* or *Referer* HTTP-request header. The module uses the algorithm of De Ryck et al. [93]. An origin O is considered as trusted to issue a cross-site request to target T, if (1) O is the same as T, or (2) O and T share the same root domain, or (3) there was an earlier delegation from T to O. Delegation from T to O means that (a) there was an earlier POST request to target O with origin T or (b) a redirect to O within the HTTP response from T.

The second CSRF module is based on the *Content Security Policy* (CSP) and *Referer* policy of modern browsers, such as Firefox and Chrome. These browsers implement a security policy mechanism that allows a server to restrict the behavior of web clients regarding the inline execution of JavaScript, the loading of JavaScript from external resources, and others. The restrictions are enforced using a special HTTP header which can contain a URL for reporting violations. This CSP header can be set to report-only, e.g., the browser only reports violations to the specified URL, but it does not enforce the rules. The module starts with a restrictive policy and the report-only attribute which is used to refine the policy based on the observed client behavior. Later the report-only mode is switched off and the policy is enforced.

Signature Analysis for Web Content

The purpose of the second step is feature extraction for the derivation/refinement of machine learning models from the domain content. Currently applied technologies for the analysis of the data content and thus the overlay structures are often based

on simple signature comparisons which are formulated by means of regular expressions that describe questionable language constructs. The use of regular expressions is problematic because the data structures of web applications cannot be described by them. If attackers manipulate the structure of web documents, as described in the Sections 4.3 and 4.4, they can bypass the analysis units, e.g., machine learning modules, that rely on input which is extracted from the document structure. Therefore, the analysis of web data structures requires context-sensitive parsers that can analyze different generations of the dominant languages. The parsers have to detect whether a document is largely correct according to the terms of the specification. In addition, they have to distinguish between format errors and analysis evasion attempts. Based on these requirements, the overlay analysis library for web documents includes parsers for a signature-based detection of evasion attempts for different languages, such as HTML/XML, CSS, and JS, independent mechanisms for checking dangerous character set encodings, and character set converters with an error analysis. The signature recognition ensures the analysis and the detection of evasion attempts, as described in section 4.3 and to some extent also evasions from section 4.4. Encoding analysis is limited to the Internet Assigned Numbers Authority (IANA) database of registered encodings[5]. It can address various problems, such as erroneous proprietary text encodings or invalid and non-printable characters.

Reactions to deviations detected in the signature unit depend on the type of the error. Deviations in the encoding, e.g., a mismatch with encoding labels of the IANA database, are blocked directly. Other potential evasions are only *recorded as additional document features* and later tested against the web application model.

Construction of Machine Learning Models for Web Applications

The last step of the overlay analysis is to assess the "style" of the web application to decide whether the active content matches it. The idea behind this is to only allow a web application to pass the firewall if its structure corresponds to the known style of the application. This is decided based on a model of the related HTML and/or JavaScript structure. In contrast to existing approaches that use models of known malicious code for the identification of unknown malicious code, we separate each allowed web application from a corpus of other web applications, e.g., the *Alexa.com* Top 500 URLs.

Feature extraction and mapping. The feature extraction approach is based on the following assumption. Almost all text-based web documents use a tree-like data structure, such as the *document object model* (DOM) of HTML or the *abstract syntax tree* (AST) of JavaScript. It is possible to enumerate known elements of a language, such

[5] `https://www.iana.org/assignments/character-sets/character-sets.xhtml`

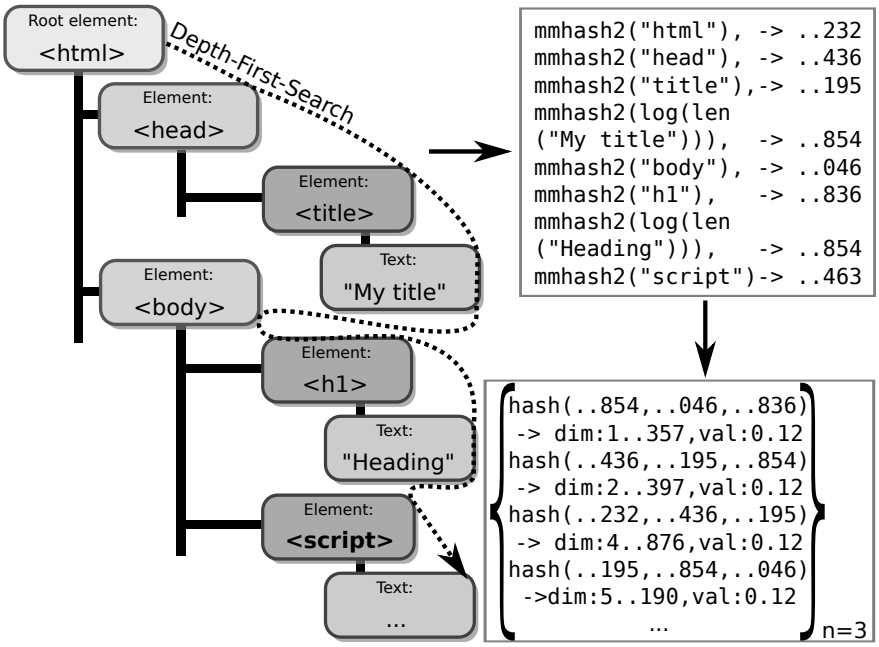

Figure 4.10: Feature extraction from the document object model

as HTML elements and attributes or JavaScript AST nodes, by means of the underlying standards, e.g., all reserved words of HTML/XML and JavaScript/EcmaScript. A depth-first search of the resulting trees with a sliding window moving over the enumerated nodes can be used to generate a set of n-grams to describe the document structure. This is exemplarily presented in Figure 4.10 for HTML. The depth-first search starts at the *html* tag. The tags and their attributes are hashed using a *Murmur Hash*[6]. For text nodes, such as headings, attribute values, and paragraphs, the text length is determined and the obtained numbers are normalized to a logarithmic scale. This first step yields a sequence of hash numbers in the order of the depth-first visit of DOM nodes. To generate models over the document structure several of the hash values have to be linked together. This is achieved by generating n-grams over the sequence of hashes, which are hashed again with the same function. The example of Figure 4.10 depicts this for the case of $n = 3$ which results in a sparsely populated feature vector with a maximum dimension of $2^{\#hashbits}$. For attack-resistant models, a larger context (larger n) is used. The hashes over the n-grams are used as indexes for the feature vector. The value of a vector dimension is the number of occurrences of the respective n-gram normalized to the l^2-norm.

For JavaScript, the same methodology is applied to the abstract syntax tree, as depicted in Fig. 4.11. The depth-first search walks through the enumerated nodes of the compiled script. Structural nodes, such as statement lists, function calls, argument lists, and argument names are extracted, but their values are ignored. The extracted

[6] `https://github.com/aappleby/smhasher`

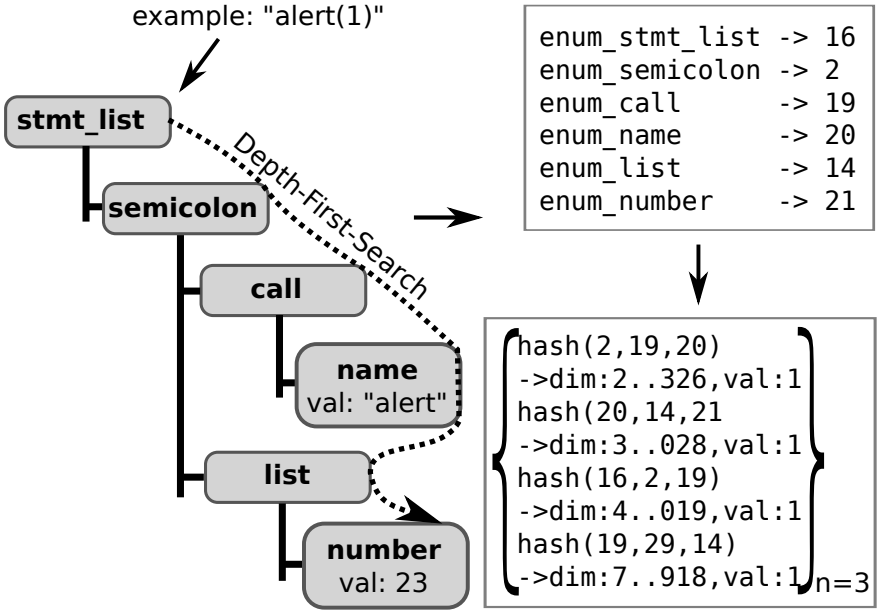

Figure 4.11: Feature extraction from the JavaScript AST

nodes are not hashed as in HTML because it is usually possible to directly extract the enumerated types from the underlying library, e.g., Mozilla SpiderMonkey[7]. However, the resulting n-grams are hashed with the same hash as for HTML. In contrast to HTML, the resulting feature vector does not track the number of occurrences of n-grams. Instead, the vector dimension is set to 1 if the corresponding n-gram is present in the data set and (implicitly) to zero otherwise (the applied libsvm[8] library can handle the index and does not require null values in sparsely populated feature vectors).

General model construction process. The web application model is based on two-class *support vector machines* (SVMs) [152] constructed from a larger set of the previously generated feature vectors. One class describes the document structure of the web application itself and the other one represents a set of other web applications which are used as counterexamples. Feature vectors from the counterexamples are randomly selected based on a fixed proportion compared to the number of vectors in the domain. Then the feature vectors of the domain are combined with the selected counterexamples to construct a linear SVM. Structurally identical fragments/pages are assigned to the domain, i.e., the intersection of the domain vector set and counter example vector set is assigned to the domain vector set when constructing the SVM.

[7] `https://developer.mozilla.org/en-US/docs/Mozilla/Projects/SpiderMonkey`
[8] `https://www.csie.ntu.edu.tw/~cjlin/libsvm/`

The process constructs several models, for example, with different ν-parameters. In the validation phase, the quality of each model is measured based on the following parameters (order = priority) to select the best model: (1) the detection accuracy for the test vectors from the domain, (2) the detection accuracy for the test vectors from the counterexamples, and (3) the number of support vectors in the model as a quality measure for its generalization. The model with the fewest remaining support vectors is used in this case. Additionally, (4) the ν-parameter of the SVM is used as a second (partially redundant) quality measure for the generalization of the model. The minimum value of ν is used in this case, because it is the target for the maximum number of misclassified training data and the maximum number of used support vectors.

Refinement of HTML models. The previously described extraction and model construction process makes it possible to distinguish web applications, such as Google and Facebook (see Figure 4.12). As it can be seen from the figure, the two web applications differ in their code size (logarithm of the code length for *onload*, *onsubmit* and *onclick* handlers), in their positioning of active code (randomly deeply nested locations for Google, such as inside *div* tags, and head of document for Facebook), as well as in further differences regarding the nesting of script tags. The process allows to identify changes of web applications that are common for stored XSS attacks, such as new child tags, that do not correspond to the order of the existing tag structure (cf. Fig. 4.1) or new attributes which were not there before.

```
Google
('body', 'onload', 3)
('div', 'onload', 1)
('center', 'div', 'script')
('body', 'center', 'script')

Facebook
('a', 'onclick', 2)
('form', 'onsubmit', 2)
('html', 'head', 'script')
('html', 'body', 'script')
```

Figure 4.12: n-grams from Google and Facebook document structures

In real-world scenarios, however, such changes would often remain undetected, since the examples in Figure 4.1 assume that the attack results in a well-formed HTML document which can be analyzed by any parser. Attacks, such as the splitting of nodes, attributes, or tags, as depicted in Figure 4.2, usually leave some fragments inside the modified document which destroy its structure. These fragments are common for reflected XSS attacks and they are interpreted ambiguously between different HTML parser implementations.

Frequent results of ambiguous interpretations of markup code inside of analysis procedures are: (1) tags that are simply ignored due to their wrong structure, (2) tags that are interpreted as attributes or attribute values, (3) tags that are interpreted as plain text, and (4) attributes that are interpreted as plain text. Therefore, the most

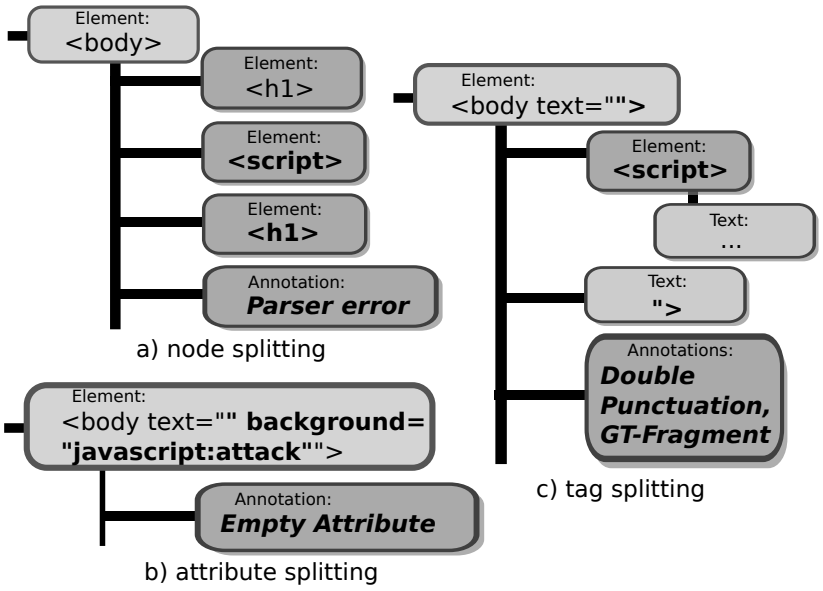

Figure 4.13: DOM annotations with identified structural problems

interesting changes do not reach the machine learning module in many cases. In order to prevent this the extracted features are augmented with additional details from the signature detection unit, such as parser warnings, identified typical attack fragments like *&gt*, *%<charcode>*, and uncommon structures, e.g., empty attribute values, as depicted in Figure 4.13. The annotated DOM nodes become a part of the extracted features and allow one to build better models which can ward off additional malicious inputs.

4.5 Implementation Details

The proposed measures have been implemented in an analysis library that comprises parsers, signature analysis modules, and machine learning methods (see Fig. 4.14). The parsers extract the basic characteristics of the web documents, as described above, and the signature units enrich the extracted features with detected inconsistencies. The overall structure is passed to the modules for machine learning. The analysis library is internally divided into parser and analysis plugins which are connected to each other, as depicted in Fig. 4.15. All parsers share a common interface, called *stream_parser*, which implements the logic of the Inspection and Modification Protocol. Analyzers are implemented using an interface, called *web_doc_analyzer*. This interface allows to enforce restrictions on the parser hierarchy, such as the connection between CSS parsers and CSS analyzers. It further enables to re-export parts of a document to other stream

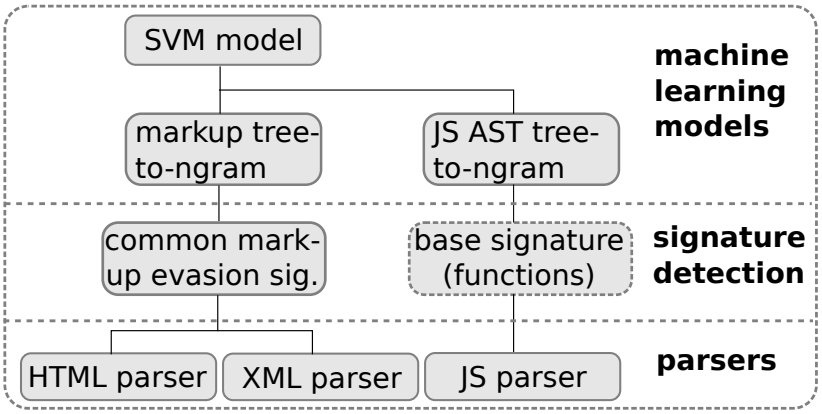

Figure 4.14: Analysis Library for Web Applications (excerpt)

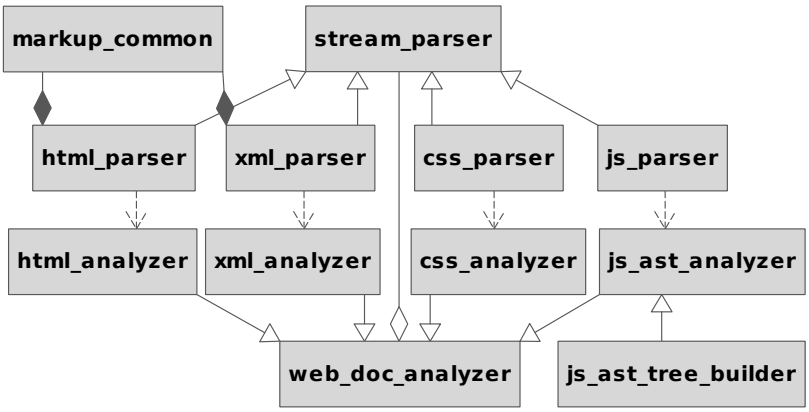

Figure 4.15: Relationship between parsers and analysis plugins

parsers, e.g., the export of HTML *script*-sections to a JavaScript parser and *style*-sections to a CSS parser. Two other special analyzers, the *js_ast_tree_builder* and a *common_markup_tree_builder*, which is encapsulated inside of a *markup common* module, export the data in a manner which is suitable for machine learning. The embeddings for the experiments in the subsequent section were accomplished with *sally*[9].

4.6 Experimental Evaluation

In order to evaluate the applicability of the approach for the identification and selective passing of web applications a part of the *Alexa.com* Top 500 Germany URLs was repeatedly mirrored and analyzed for a period of 15 days. Note that the Alexa Top 500

[9] `http://www.mlsec.org/sally/download.html`

global list cannot be used for such an experiment because it has not the same diversity as the country-specific top lists, since the global top list includes repeated country-specific versions of Google and other major sites. Out of the Alexa URLs, 253 were directly reachable, while most of the other ones represented interfaces of advertisement networks which blocked the access due to missing *referer* headers and about another 130 were not covered by the web crawler because they diverted the HTTP requests to another domain. The first two experiments are based on an (unauthenticated) extraction of the web documents from the respective sites with a link depth of 6 limited to 60 seconds extraction time, resulting in 27,456 to 30,829 HTML documents and 1,878 to 1,975 JavaScript documents per day.

The first experiment evaluated the URLs from the first two days against each other to analyze the capability of the generated models to identify individual web applications and to fix the model parameters (e.g., n-gram size). The second experiment tested the stability of the generated models over the full period. A third experiment evaluated the applicability of the approach to ward off (potentially) malicious input. For this, 4386 HTML documents and 400 JavaScript documents were evaluated, which were extracted from the following security frameworks: WebScarab[10], Vega[11], arachni[12], beef[13], grabber[14], grendel[15], metasploit[16], nikto[17], sqlmap[18], w3af[19], wapiti[20], and zap[21]. The final experiment evaluated the performance of the overall system.

Evaluation Criteria

To evaluate the approach various criteria were measured based on *training sets (T)* extracted during the first day for each domain which consisted of the feature vectors from the HTML pages and JavaScript fragments of a domain as well as randomly selected feature vectors from the other domains as counterexamples. The combined domain and counterexample feature vectors were used to train the support vector machines for each domain which are based on a linear kernel. The models are evaluated against several *validation and test sets (V/T)* which consisted of the feature vectors and counterexamples for each domain that were extracted and generated during the subsequent

[10] `https://www.owasp.org/index.php/Category:OWASP_WebScarab_Project`
[11] `https://subgraph.com/vega/`
[12] `http://www.arachni-scanner.com/`
[13] `http://beefproject.com/`
[14] `https://github.com/neuroo/grabber`
[15] `https://sourceforge.net/projects/grendel/`
[16] `https://www.metasploit.com/`
[17] `https://cirt.net/Nikto2`
[18] `http://sqlmap.org/`
[19] `http://w3af.org/`
[20] `http://wapiti.sourceforge.net/`
[21] `https://github.com/zaproxy/zaproxy`

days. *True-positives (TPs)* indicate the domain vectors that are classified as domain vectors. *True-negatives (TNs)* characterize vectors from the counterexamples in each validation/test set that are classified as other domain vectors. *False-positives (FPs)* identify vectors from the counterexamples in each validation/test set that are classified as domain vectors. *False-negatives (FNs)* measure domain vectors that are classified as other domain vectors.

Based on these four basic measures, other metrics are defined to identify the best models or model parameters. The *true-positive rate (TPR)* indicates the proportion of the domain examples that the model predicts correctly: *TP/(TP+FN)*. The *true-negative rate (TNR)* specifies the proportion of the counter examples of the validation/test set that the model predicts correctly: *TN/(TN+FP)*. The *false-positive rate (FPR)* defines the proportion of the counter examples of the validation/test set that the model predicts falsely as domain code: *FP/(FP+TN)*. The *false-negative rate (FNR)* is the proportion of the domain examples that the model misinterprets as other domain code: *FN/(FN+TP)*. The *accuracy (A)* specifies the proportion of the validation/test set that the model predicts correctly: *(TP+TN)/(TP+TN+FP+FN)*. The *error-rate (E)* defines the proportion of the validation/test set that the model predicts incorrectly: *(FP+FN)/(TP+TN+FP+FN)*. The *precision (P)* indicates the proportion of the positive domain examples that were really positive: *TP/(TP+FP)*. The *F-measure (F1)* is the harmonic mean of the precision and the true-positive rate: *2*P*TPR/(P+TPR)*.

Applicability for Generating Web Application Models

Experiment 1 evaluated the capability of the generated models to *identify* individual web applications. For this, the vectors of the validation set of each domain were evaluated against randomly selected vectors from all other domains. The feature extraction process for the vectors uses different *n-gram* values to identify how much context is needed for a model with a good detection accuracy. The outcome of this experiment is summarized in Tables 4.1 (for HTML) and 4.2 (for JavaScript). For HTML, the value $n = 7$ yields the best accuracy (A), precision (P), and F-measure (F1) as well as the lowest error-rate (E) for the resulting models. To increase the capability of the firewall to reliably identify web applications, the JavaScript fragments were considered as well. For this, the capability of the generated models to identify JavaScript fragments as part of the corresponding web applications were evaluated. The outcome of this evaluation is summarized in Table 4.2. The best model for JavaScript is achieved for $n = 8$. Compared to the HTML models the JavaScript models have less predictive capabilities. However, the models for the JavaScript fragments are not meant to be used alone because the proof of domain membership is not obtained from a JavaScript fragment alone. Instead the result is combined with the result of the surrounding HTML document. Table 4.3 represents the combined detection capability of the best

n	TPR [%]	TNR [%]	FPR [%]	FNR [%]	A [%]	E [%]	P [%]	F1 [%]
3	98.29	99.18	0.82	1.71	99.08	0.92	93.77	95.98
4	98.77	99.57	0.43	1.23	99.48	0.52	96.65	97.69
5	99.16	99.7	0.3	0.83	99.6	0.4	97.33	98.24
6	**99.4**	99.72	0.28	**0.6**	99.68	0.32	97.79	98.56
7	**99.4**	**100**	**0**	**0.6**	**99.93**	**0.07**	**100**	**99.7**
8	99.22	99.75	0.25	0.78	99.69	0.31	98.01	98.61
9	99.2	99.79	0.21	0.80	99.72	0.28	98.3	98.74

Table 4.1: Detection capability of HTML models

n	TPR [%]	TNR [%]	FPR [%]	FNR [%]	A [%]	E [%]	P [%]	F1 [%]
3	98.84	75.9	24.1	1.16	78.44	21.56	33.89	50.47
4	99.0	78.6	21.40	1.0	80.86	19.14	36.64	53.48
5	**99.2**	83.23	16.77	**0.81**	85.0	15.0	42.51	59.51
6	99.06	86.14	13.86	0.94	87.58	12.42	47.19	63.92
7	99.01	87.29	12.71	0.99	88.59	11.41	49.33	65.85
8	98.82	**90.79**	**9.21**	1.18	**91.68**	**8.32**	**57.28**	**72.52**
9	98.8	90.49	9.51	1.2	91.42	8.58	56.5	71.89

Table 4.2: Detection capability of JavaScript models

HTML and best JavaScript models. A firewall that removes JavaScript fragments if the HTML model and the JavaScript model determines the code as third-party code would detect 94.67% of the foreign code (TNR) and mistakenly remove about one out of hundred fragments (FNR).

Experiment 2 evaluated the stability of the models over a period of 13 days. The period resulted from the previous experiment. The complete data set encompasses 15 days. The data sets of the first day were already used in the first experiment to generate the domain models. From the resulting model set, the parameters for the best models were fixed with the help of the data sets of the second day. Accordingly, the data sets of the remaining 13 days were evaluated against the models from the first day to test the stability of these models. The results of this experiment are shown in Tables 4.4 (HTML), 4.5 (for JavaScript) and 4.6 (combined models). The relevant value is the development of the error rate (E). If there is an increase then the model

TPR [%]	TNR [%]	FPR [%]	FNR [%]	A [%]	E [%]	P [%]	F1 [%]
99.06	**94.67**	5.33	**0.94**	95.16	4.84	69.9	81.96

Table 4.3: Combined detection capability

day	TPR [%]	TNR [%]	FPR [%]	FNR [%]	A [%]	E [%]	P [%]	F1 [%]
1	**99.14**	99.64	0.36	0.86	99.58	**0.42**	97.14	98.13
2	**99.08**	99.64	0.36	0.92	99.58	**0.42**	97.19	98.12
3	**98.96**	99.67	0.33	1.04	99.59	**0.41**	97.39	98.17
4	**98.7**	99.66	0.34	1.3	99.56	**0.44**	97.35	98.02
5	**98.88**	99.63	0.37	1.12	99.54	**0.45**	97.08	97.97
6	**99.04**	99.64	0.36	0.96	99.58	**0.42**	97.2	98.11
7	**98.68**	99.66	0.34	1.32	99.55	**0.44**	97.35	98.01
8	**98.7**	99.63	0.37	1.3	99.53	**0.47**	97.12	97.9
9	**98.52**	99.65	0.35	1.48	99.52	**0.48**	97.22	97.86
10	**98.53**	99.67	0.33	1.46	99.54	**0.46**	97.36	97.94
11	**98.51**	99.64	0.36	1.49	99.52	**0.48**	97.19	97.84
12	**98.08**	99.58	0.42	1.92	99.41	**0.59**	96.67	97.37
13	**98.09**	99.63	0.37	1.91	99.46	**0.54**	97.04	97.56

Table 4.4: Detection stability of HTML models

may need to be re-trained regularly. For the analysis of HTML, there seems to be a slight increase (0.12%), while the JavaScript model seems to be stable within its accuracy. The main reason for the increasing error rate of HTML appears to be the drop in the true-positive rate of around 1%. This could be a result of too little data in the initial page extraction. The model possibly stabilizes when the time window for the initial mirroring of the domains is extended.

Experiment 3 evaluated the capability of the models to ward off malicious input. For this purpose, the counterexamples to the domains in the data sets of the third day were replaced by extracted feature vectors of HTML and JavaScript fragments of various web security frameworks. Most of the superseded vectors were based on benign auxiliary code of the web security scanners which is necessary to set up the attacks, but some of these vectors contained real attacks against the web browsers. Regardless of this fact, all these vectors do not belong to the respective domain and accordingly the response of the model to the changed input is tested. The results of the experiment are summarized in Table 4.7. The HTML models recognize 99. 97% (TNR) of the HTML documents as someone else's code, while the JavaScript (*JS*) models reject 93.43% of the JavaScript fragments as foreign code. If the results of the two models are combined (*COMB*) 99.6% of the foreign code would be rejected.

Combined results and comparison with other approaches. This section summarizes the results of the previous experiments, and compares them with other approaches to detect web applications and malicious code. The upper part of table 4.8 summarizes the results of the own experiments. The first line (*HTML*) summarizes the results of

day	TPR [%]	TNR [%]	FPR [%]	FNR [%]	A [%]	E [%]	P [%]	F1 [%]
1	98.21	90.74	9.26	1.79	91.57	**8.43**	56.99	72.13
2	97.45	90.84	9.16	2.55	91.58	**8.42**	57.09	72.0
3	97.48	90.82	9.17	2.52	91.56	**8.43**	57.05	71.98
4	96.94	90.74	9.26	3.05	91.43	**8.57**	56.69	71.54
5	97.5	90.77	9.23	2.5	91.52	**8.48**	56.9	71.86
6	97.76	90.77	9.23	2.24	91.55	**8.45**	56.97	71.99
7	97.72	90.82	9.18	2.28	91.59	**8.41**	57.1	72.08
8	97.43	90.88	9.12	2.57	91.61	**8.39**	57.19	72.08
9	96.98	90.9	9.1	3.02	91.58	**8.42**	57.12	71.89
10	97.1	90.92	9.08	2.9	91.61	**8.39**	57.2	72.0
11	97.03	90.83	9.16	2.97	91.52	**8.48**	56.96	71.78
12	97.08	90.96	9.04	2.9	91.64	**8.36**	57.3	72.06
13	96.55	90.96	9.04	3.45	91.58	**8.42**	57.17	71.82

Table 4.5: Detection stability of JavaScript models

day	TPR [%]	TNR [%]	FPR [%]	FNR [%]	A [%]	E [%]	P [%]	F1 [%]
1	98.6	94.54	5.45	1.4	95.0	**5.0**	69.32	81.41
2	98.16	94.69	5.3	1.84	95.08	**4.92**	69.81	81.59
3	98.14	94.73	5.27	1.86	95.11	**4.89**	69.96	81.68
4	97.71	94.66	5.34	2.29	95.0	**5.0**	69.56	81.27
5	98.1	94.66	5.33	1.89	95.05	**4.95**	69.69	81.49
6	98.33	94.68	5.3	1.67	95.09	**4.91**	69.81	81.65
7	98.14	94.72	5.28	1.86	95.1	**4.9**	69.9	81.65
8	97.98	94.71	5.29	2.02	95.07	**4.93**	69.82	81.54
9	97.65	94.72	5.28	2.35	95.05	**4.95**	69.82	81.42
10	97.74	94.8	5.2	2.26	95.12	**4.88**	70.12	81.66
11	97.68	94.73	5.27	2.32	95.06	**4.94**	69.84	81.45
12	97.51	94.69	5.31	2.49	95.0	**5.0**	69.65	81.26
13	97.23	94.77	5.23	2.77	95.04	**4.95**	69.92	81.34

Table 4.6: Detection stability of the combined models

	TPR[%]	TNR[%]	FPR[%]	FNR[%]	A[%]	E[%]	P[%]	F1[%]
HTML	99.13	**99.97**	0.03	0.87	99.96	0.04	98.27	98.7
JS	98.21	**93.43**	6.57	1.79	94.71	5.28	84.62	90.91
COMB	98.6	**99.6**	0.4	1.4	99.57	0.43	89.96	94.08

Table 4.7: Malware detection capability (Malware = TNR)

	TPR[%]	TNR[%]	FPR[%]	FNR[%]	A[%]	E[%]	P[%]	F1[%]
HTML	**98.76**	99.77	0.23	**1.24**	**99.69**	**0.31**	97.41	**98.08**
JS	97.48	90.9	9.1	2.52	91.66	8.34	58.33	72.99
COMB	98.04	95.65	4.35	1.96	95.88	4.12	70.83	82.24
Zozzle1	98.74	98.48	1.52	1.26	98.51	1.49	88.23	93.19
Zozzle2	90.8	**100.0**	**0.0**	9.2	99.2	**0.8**	**100.0**	95.18
Zozzle2*					99.06	0.94		

Table 4.8: Combined results compared to related approaches

the best model ($n = 7$) for HTML from table 4.1, as well as all HTML results of table 4.4 and the first row (HTML malware results) of table 4.7. The second line (*JS*) summarizes the results of the best model (n = 8) for JavaScript from table 4.2, as well as all JavaScript results in table 4.5 and the second row (JavaScript malware results) of table 4.7. The third line (*COMB*) summarizes all results of tables 4.3 and 4.6, as well as the third row (combined malware results) from table 4.7.

There are very few other approaches for the detection of web applications that expose all relevant results in an evaluation. The authors of SWAP [85], for example, test an unknown number of attacks against three web applications (claiming 100% detection accuracy) without making a false-positive analysis. Rui Wang et al. [142] indicate for one experiment the resulting precision, recall, and F-measure and for another one the true-positive and false-negative rates, but they do not report any false-positive-related measure at all. The same applies to the approach of Ismail et al. [140]. Johns et al. can detect 100% of the reflected XSS attacks in their experiments [148], but their false positive analysis is incomprehensible because they are merely referring to two distribution tables without any summary. The author of JaSPIn [92] performs two different experiments. In one experiment, he tested 59 exploits against a web application in which he achieved a 100 percent detection rate, but he reveals no false positives for this experiment. In another experiment, he tested 60 web application models for false-negatives (although he referred to these as false-positives). The appropriate models can assign around 80% of the JavaScript calls to the respective web applications and fail at the rest. Only for Asynchronous JavaScript and XML (AJAX)-based applications, a recognition accuracy of 91% is achieved, which is probably connected to the regularly repeating function call patterns of AJAX.

Zozzle is the only approach, for which all evaluation values can be calculated (at least for one experiment). Zozzle recognizes exactly one XSS-based attack (Heap Spraying) based on a set of 919 exploits. For the first false-positive analysis, apparently all exploits are used as training data and the resulting model is tested against a set of 7976 (benign) JavaScript contexts from the Alexa Top 50. The results of this experiment are shown in the last three rows of Table 4.8. Zozzle1 evaluated the whole set (malware + Alexa Top 50) based on hand-selected features. Zozzle2 repeated the

experiment with a χ^2-based feature selection. The missing evaluation values of the Zozzle paper were determined in the following way: $TPR = 100 - FNR; TNR = 100 - FPR; E = 100 - A$. For P and the F-measure ($F1$) the absolute values for TP and FP were calculated based on the given numbers (malware = 919; benign = 7976). Using this calculation, the results for the accuracy A and error rate E for the model based on hand-selected features (Zozzle1) could be confirmed. In reviewing the results of the model Zozzle2, a small discrepancy was found. The penultimate line (Zozzle2) of Table 4.8 reflects the published values, while the last line (Zozzle2*) was calculated based on the absolute values.

The approach proposed in this chapter is based on whitelisting domain code (using a SVM model). Compared with the only other similar approach with published results (JaSPIn) the proposed SVM model should be preferred for such a use case. Zozzle, on the other hand, is based on the blacklisting of known malicious code. For the case of heap-spraying attacks it should preferred against an SVM-based option.

Performance of the firewall architecture. The models for the web applications are generated offline. During the crawl, within 60 seconds, a sufficiently large quantity of documents (on average 109 HTML pages and 7 JavaScript documents per domain) was extracted. In the prototype of the proposed approach, up to 38 models with varying ν-parameters were generated for each domain. On the Core 2 Quad Q9550@2.83GHz test system, it took on average 6 seconds to generate these models and to identify the final domain model.

In relation to the online part of the analysis, it is important to distinguish between two cases: there is (1) a new document with a different structure, which should be assessed in relation to the previous model, and (2) an existing document with changed content but the same structure, which should be detected again. In both cases the document has to be parsed. With the current prototype, parsing a HTML document takes – depending on the complexity – between 1.3 and 314 milliseconds. Parsing a JavaScript document depends on the document size and takes between 0.4 milliseconds (a few bytes) and 1.5 seconds (1 MByte). For each document, a structural hash is made when parsing. On the base of these hashes, it can be checked in the second case whether there is already a document rating which is then used instead of a check against the model. The first case requires a feature extraction and a test against the model. The extraction time of the features is also dependent on the document size. For HTML, the feature extraction requires between 10 (8 KByte document) and 390 milliseconds (12 MByte document). For JavaScript, the feature extraction requires between 10 (4 KByte document) and 440 milliseconds (8 MByte document). The validation time against the model is relatively constant and requires for HTML between 60 and 70 milliseconds and for JavaScript between 20 and 30 milliseconds. Note that the parser of the prototype is extremely inefficient because it contains a

lot of number-to-text conversions for the implementation of the experiments, and improvements are relatively easy to implement.

4.7 Conclusions

This chapter investigated the problems of perimeter firewalls when analyzing web traffic. A firewall architecture has been proposed that addresses the entire process chain starting from the data transfer with the HTTP protocol, which is prone to evasion attacks, via the analysis of manipulated web documents to the extraction and analysis of active content. The last step of the firewall analysis assesses the style of the web application using machine-learning methods. The basic idea is to allow a restricted set of web applications to pass the firewall based on a model of their HTML and JavaScript structure and to remove active content from the other web sites which are not part of this set. Promising results were achieved when evaluating the capability of the resulting models to identify the underlying web applications and their ability to enforce their structure when confronted with additional malicious input.

Based on the results of the evaluations, still 1 out of 116 HTML pages and 1 out of 56 JavaScript fragments may be classified as non-domain code by a domain model. This is an issue for a firewall approach, since it removes the respective active code from the page with a possible loss of functionality. It was discussed to combine HTML and JavaScript models for further evidence of domain membership to solve a part of this problem. The proposed method has, however, more degrees of freedom for determining the meaningfulness of a statement about the similarity of a feature vector to a class, which can be used in future work to improve the detection accuracy.

Distance-based measurement. Since the applied SVMs are based on linear kernels, there may be other means to gain further knowledge from the individual models. The idea is to measure the distance of test vectors to the hyperplane of the model to obtain some insights about the reliability of the prediction. The further the test vector is away from the hyperplane the more it should be attributed to the corresponding class. In uncertain cases in which the test vector is too close, the firewall should let the document/fragment simply pass.

Direct use of extracted features. Known benign HTML/JavaScript is checked in this case, not as an n-gram against the model, but kept directly as an n-gram "signature". For minor changes (e.g., only one modified n-gram in a chain of n-grams), a simpler metric, such as the edit distance, may be more meaningful as a measure for class similarity.

Separation of benign and malicious counterexamples. Currently the SVM models of the web applications include only two classes that either assign the tested code samples to the corresponding web page or determine it as foreign code. Optionally,

another SVM that tests code fragments against known malicious code could further increase the detection accuracy.

Client-side analysis. In the IceShield approach [141], the malware analysis library was sent directly as JavaScript (included/embedded in the web page) to the client side and the necessary analyzes were performed in the browser. This approach initially raised questions with regard to the self-protection of the analysis library against attack code that runs in the same context within the transferred web page. This problem was solved by the IceShield developers by utilizing a JavaScript feature, called *Object.defineProperty(),* to freeze the *configurable* object properties of overwritten methods used in the context of the library. If the linear decision function of the IceShield approach would be replaced by the linear SVM used in this approach, both approaches could be integrated into each other. The SVM library *libsvm*[22] may be translated with *emscripten*[23] into the *asm.js*[24] format which allows a direct use of (translated) C/C++ code in JavaScript programs. HTML 5 web storage[25] can serve as storage for the models described here.

[22] `https://www.csie.ntu.edu.tw/~cjlin/libsvm/`
[23] `https://kripken.github.io/emscripten-site/`
[24] `http://asmjs.org/spec/latest/`
[25] `https://www.w3.org/TR/webstorage/`

5 Summary and Outlook

In order to be able to further address the increasing threats to the IT security of complex networks, a mix of preventive and reactive security mechanisms is required. A glance at the area of the better defined safety engineering in Chapter 1 shows that the solution of complex IT security problems requires a systematic security engineering process that encompasses all potentially vulnerable components. Therefore, network security problems require a security engineering process that encompasses the entire network, in particular also the local area networks as indicated in the introduction of this thesis. The focus should initially be laid on preventive measures that prevent known attacks rather than recognize them later. However, since there will still be attacks which cannot be prevented with reasonable effort, reactive technologies, such as intrusion detection systems, are also of importance.

The first part of this thesis already proposed initial steps of the security engineering process, e.g., the description of IT systems and networks from the attacker perspective. The next steps should always include the analysis of known attacks and preventive defense measures (mitigation strategies) against these attacks. A significant similarity of known attacks is, for example, the spreading of malicious code into internal networks and the extraction of data from compromised subnets and hosts. Chapter 2 started accordingly with an analysis of well-known and lesser-known attacks on local area networks. As part of a potential mitigation strategy against these attacks, software-defined networking (SDN) was applied as a vehicle for centralizing information about all network activities in a central authority – the SDN controller – that manages all network connections and hence the associated data flows. The SDN technology allows to provide networks with security services that perform basic tasks, such as address configuration, address resolution, and firewalling within the network, much more efficient than on scattered individual systems. The resulting secure networks are based on switched routing which uses auxiliary information from the address configuration services to enforce a strong binding between a packet and its origin as well as its target that makes attacks very difficult.

The second part of the security engineering process includes the provision of technologies that can detect known attacks. This functionality is already covered by existing network intrusion detection systems (NIDS), such as Snort. Single-threaded NIDS were originally designed for a performance of a few ten to one hundred megabits per second. Highly variable communication relations and constantly increasing network bandwidths, however, more frequently force NIDS to handle high peak rates which

often reach multiple gigabits per second. Accordingly, parallelization strategies are necessary to be able to apply intrusion detection technologies within a network, e.g., on the backbone. Based on an analysis of known parallel NIDS, such as SURICATA, as well as own prototypes of various design options, Chapter 3 developed and evaluated a parallel NIDS architecture that can withstand these changing traffic conditions. It maintains a high throughput and a thorough analysis of all security-relevant data.

The first route of infection through malicious email attachments and compromised web applications will remain a special issue of intrusion detection processes for the near future. As a possible mitigation strategy Chapter 4 proposes a novel concept for a client-side/server-side web firewall architecture with an additional analysis library that extends the firewall analysis by capabilities to normalize web traffic at the application layer, to detect firewall evasions, and to classify web applications using machine learning methods with the objective to identify the web applications and to selectively pass them according to the given firewall policy. If this technology can be transferred to the end systems, i.e., into the browsers or into an analysis library for generated web pages on the server side, the cost of web attacks would increase enormously.

Of course, not all IT security issues can be solved within the framework of a single thesis. For the foreseeable future, the following issues need to be addressed.

Dealing with malicious mail attachments. A popular entry point for the initial infection of computers is the delivery of malicious code via email attachments. When writing this thesis, the proportion of malicious code in the form of ransomware increased enormously. The currently popular business model is to encrypt files on the compromised computer and then extort money for their decryption. This goes back to the fact that Bitcoin money can be transferred anonymously to blackmailers in a barely traceable manner. The early versions of blackmail trojans used clearly recognizable executable files, such as *.exe*. Later attackers partially resorted to lesser-known program extensions, such as *.com* and *.scr*, in order to conceal the active nature of the code. Now the concealment measures go so far that even JavaScript[1] can contain executable code (for the Windows Script Host) and also the macro viruses have returned[2]. The latter even work if the (*.docm*, *.dotm*) files were renamed into apparently innocuous files (e.g., *.rtf* – rich text files). The detection of such malicious code requires, therefore, a deep inspection of the document containers. Office-Makros can still be detected using rule-based processing by determining whether the [Content_types].xml document refers to the MIME type *application/vnd.ms-word.document.macroEnabled.main+xml* (+ macro templates). In other cases, machine learning methods are needed to distinguish malicious code from benign code. A variation of the process developed in Chapter 4 can be used under certain circumstances to distinguish malicious JavaScript and other document

[1] `https://isc.sans.edu/diary/Locky%3A+JavaScript+Deobfuscation/20749`
[2] `http://blog.talosintel.com/2016/08/macro-intruders-sneaking-past-office.html`

structures from benign structures of the same type. Executable code in .exe files, however, requires new detection methods. Interesting features in the analysis of (malicious) executable files are, for example, the presence or absence of digital certificates, encrypted sections, the use of packers, the number of imported symbols (functions), and the presence of known anti-debugging routines.

Monitoring of encrypted traffic. The increasing use of traffic encryption prevents the application of traditional intrusion detection analyses. Neither the deep packet inspection of NIDS nor classical flow aggregation can be applied to encrypted traffic. Novel statistical methods are needed to at least roughly classify and analyze this traffic. A cryptographic analysis could, for example, use statistical methods to develop models based on empirical data that allow a classification of encrypted traffic, e.g., with respect to the underlying applications and cryptographic libraries, and extract other additional features, such as the applied ciphersuites from the cryptographic handshakes, to detect anomalies. In recent years, the focus in the security analyses, for instance, increasingly shifted to implementations of cryptographic standards. Many problems were discovered, such as the revealing of client-[3] and server-side[4] private keys or the derivation of session keys[5]. In one case, even a downgrade of encrypted channels to plaintext transfers was possible[6]. Since the cryptographic handshake is in plain text, features like the applied crypto protocol version, negotiated ciphersuites, server and client identities, prime parameters, and the traversed states until the completion of the handshake could be extracted and provided to machine learning techniques to detect the related anomalies.

Cyber-physical systems security. Cyber-physical systems are characterized by a highly heterogeneous IT composed of standardized components and proprietary solutions which interact with the physical world in a potentially safety-hazardous manner. Examples of cyber-physical systems are critical infrastructures, such as power plants and energy distribution networks. The detection of vulnerabilities and attacks in such an environments involves a number of challenges that cannot be solved with classical IT security concepts. The smooth transition to the physical world, for example, involves a process-related IT with sensors and actuators that is driven by a growing number of rare and unknown communication protocols, which are not nearly as robust and secure as standardized protocols. In order to cope with these issues specialized systems are required that can monitor the traffic of the respective networks in a phase of self-adjustment to deduce rules and models for normal operation, and to observe and to reproduce the state

[3] https://www.qualys.com/2016/01/14/cve-2016-0777-cve-2016-0778/openssh-cve-2016-0777-cve-2016-0778.txt

[4] http://heartbleed.com/

[5] http://seclists.org/oss-sec/2016/q1/261 + https://drownattack.com/

[6] https://mitls.org/pages/attacks/SMACK

transitions of unknown protocols. The derived transport and protocol rules can then be used to detect abnormalities.

Bibliography

[1] TM-5-698-4, "Failure Modes, Effects and Criticality Analyses (FMECA) for Command, Control, Communications, Compute, Intelligence, Surveillance, and Reconnaissance (C4ISR) Facilities," Department of the Army, Tech. Rep., 9 2006.

[2] Mandiant, "APT1: Exposing One of China's Cyber Espionage Units," http://intelreport.mandiant.com/Mandiant_APT1_Report.pdf, 2013.

[3] Intel Corporation, *Intel®64 and IA-32 Architectures Software Developer's Manual*, Intel Corporation, P.O. Box 5937, Denver, Colorado 80217-9808, September 2015.

[4] MIL-STD-1629A, "Procedures for Performing a Failure Mode, Effects and Criticality Analysis," Department of Defense, Tech. Rep., 11 1980.

[5] A. L. Martensen and R. W. Butler, "TM-89098 – The Fault-Tree Compiler," http://www.ntrs.nasa.gov/search.jsp?R=19870011332, NASA Langley Research Center, Tech. Rep., 1 1987.

[6] E. G. Amoroso, *Intrusion Detection: An Introduction to Internet Surveillance, Correlation, Trace Back, Traps, and Response.* Intrusion.Net Books, Feb. 1999. [Online]. Available: http://www.worldcat.org/isbn/0966670078

[7] N. Falliere, L. O. Murchu, and E. Chien, "W32.Stuxnet Dossier," https://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/w32_stuxnet_dossier.pdf, 2011.

[8] MITRE, "CVE-2010-2568," https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-2568, 2010.

[9] M. Graeber, "Abusing Windows Management Instrumentation (WMI) to Build a Persistent, Asyncronous, and Fileless Backdoor," https://www.blackhat.com/docs/us-15/materials/us-15-Graeber-Abusing-Windows-Management-Instrumentation-WMI-To-Build-A-Persistent2015.

[10] J. King and K. Lauerman, "Layer 2 Attacks and Mitigation Techniques for the Cisco Catalyst 6500 Series," http://www.cisco.com/c/en/us/products/collateral/switches/catalyst-6500-series-switches/white_paper_c11_603839.pdf, 2010.

[11] E. Buchanan, R. Roemer, H. Shacham, and S. Savage, "When Good Instructions Go Bad: Generalizing Return-Oriented Programming to RISC," in *ACM Conference on Computer and Communications Security, CCS Alexandria, Virginia, USA*, P. Ning, P. F. Syverson, and S. Jha, Eds. ACM, 2008, pp. 27–38. [Online]. Available: http://doi.acm.org/10.1145/1455770.1455776

[12] H. Shacham, "The Geometry of Innocent Flesh on the Bone: Return-into-libc without Function Calls (on the x86)," in *ACM Conference on Computer and Communications Security, CCS 2007, Alexandria, Virginia, USA*, P. Ning, S. D. C. di Vimercati, and P. F. Syverson, Eds. ACM, 2007, pp. 552–561. [Online]. Available: http://doi.acm.org/10.1145/1315245.1315313

[13] C. Cowan, "StackGuard: Automatic Adaptive Detection and Prevention of Buffer-Overflow Attacks," in *7th USENIX Security Symposium, San Antonio, TX, USA*, A. D. Rubin, Ed. USENIX Association, 1998. [Online]. Available: https://www.usenix.org/conference/7th-usenix-security-symposium/stackguard-automatic-adaptive-detection-and-prevention

[14] W. K. Robertson, C. Krügel, D. Mutz, and F. Valeur, "Run-time Detection of Heap-based Overflows," in *17th Conference on Systems Administration (LISA 2003), San Diego, California, USA*, Æ. Frisch, Ed. USENIX, 2003, pp. 51–60. [Online]. Available: http://www.usenix.org/publications/library/proceedings/lisa03/tech/robertson.html

[15] "Address Space Layout Randomization," https://pax.grsecurity.net/docs/aslr.txt, PaX Team, 2003.

[16] N. Ludd and A. Gabert, "Introduction to Position Independent Code," https://wiki.gentoo.org/wiki/Hardened/Introduction_to_Position_Independent_Code, Gentoo, 2015.

[17] M. Abadi, M. Budiu, Ú. Erlingsson, and J. Ligatti, "Control-flow integrity principles, implementations, and applications," *ACM Transactions on Information and System Security (TISSEC)*, vol. 13, no. 1, 2009. [Online]. Available: http://doi.acm.org/10.1145/1609956.1609960

[18] V. Kuznetsov, L. Szekeres, M. Payer, G. Candea, R. Sekar, and D. Song, "Code-Pointer Integrity," in *11th USENIX Symposium on Operating Systems Design and Implementation, OSDI '14, Broomfield, CO, USA*, J. Flinn and H. Levy, Eds. USENIX Association, 2014, pp. 147–163. [Online]. Available: https://www.usenix.org/conference/osdi14/technical-sessions/presentation/kuznetsov

[19] H. E. Petersen and R. Turn, "System implications of information privacy," in *American Federation of Information Processing Societies: Proceedings of the AFIPS '67 Spring Joint Computer Conference, Atlantic City, New Jersey, USA*, 1967, pp. 291–300. [Online]. Available: http://doi.acm.org/10.1145/1465482.1465526

[20] R. R. Linde, "Operating system penetration," in *American Federation of Information Processing Societies: 1975 National Computer Conference, Anaheim, CA, USA*, ser. AFIPS Conference Proceedings, vol. 44. AFIPS Press, 1975, pp. 361–368. [Online]. Available: http://doi.acm.org/10.1145/1499949.1500018

[21] J. P. Anderson, "Computer Security Threat Monitoring and Surveillance," James P. Anderson Co, Fort Washington, PA, Tech. Rep., 1980.

[22] D. E. Denning, "An Intrusion-Detection Model," *IEEE Transactions on Software Engineering*, vol. 13, no. 2, pp. 222–232, 1987. [Online]. Available: http://doi.ieeecomputersociety.org/10.1109/TSE.1987.232894

[23] T. F. Lunt and R. Jagannathan, "A prototype real-time intrusion-detection expert system," in *1988 IEEE Symposium on Security and Privacy, Oakland, California, USA*. IEEE Computer Society, 1988, pp. 59–66. [Online]. Available: http://dx.doi.org/10.1109/SECPRI.1988.8098

[24] S. Smaha, "Haystack: an intrusion detection system," in *Aerospace Computer Security Applications Conference, 1988., Fourth*, Dec 1988, pp. 37–44.

[25] D. Winer, "XML-RPC Specification," http://www.xmlrpc.com/spec, 1999.

[26] "Simple Object Access Protocol (SOAP) 1.2," http://www.w3c.org/TR/soap/, World Wide Web Consortium, 2003.

[27] J. Postel, "Internet Protocol," RFC 791 (INTERNET STANDARD), Internet Engineering Task Force, Sep. 1981, updated by RFCs 1349, 2474, 6864. [Online]. Available: http://www.ietf.org/rfc/rfc791.txt

[28] D. Plummer, "Ethernet Address Resolution Protocol: Or Converting Network Protocol Addresses to 48.bit Ethernet Address for Transmission on Ethernet Hardware," RFC 826 (INTERNET STANDARD), Internet Engineering Task Force, Nov. 1982, updated by RFCs 5227, 5494. [Online]. Available: http://www.ietf.org/rfc/rfc826.txt

[29] R. Droms, "Dynamic Host Configuration Protocol," RFC 2131 (Draft Standard), Internet Engineering Task Force, Mar. 1997, updated by RFCs 3396, 4361, 5494, 6842. [Online]. Available: http://www.ietf.org/rfc/rfc2131.txt

[30] J. Postel, "Internet Control Message Protocol," RFC 792 (INTERNET STANDARD), Internet Engineering Task Force, Sep. 1981, updated by RFCs 950, 4884, 6633, 6918. [Online]. Available: http://www.ietf.org/rfc/rfc792.txt

[31] S. Deering and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification," RFC 2460 (Draft Standard), Internet Engineering Task Force, Dec. 1998, updated by RFCs 5095, 5722, 5871, 6437, 6564, 6935, 6946, 7045, 7112. [Online]. Available: http://www.ietf.org/rfc/rfc2460.txt

[32] A. Conta and S. Deering, "Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification," RFC 2463 (Draft Standard), Internet Engineering Task Force, Dec. 1998, obsoleted by RFC 4443. [Online]. Available: http://www.ietf.org/rfc/rfc2463.txt

[33] R. Droms, J. Bound, B. Volz, T. Lemon, C. Perkins, and M. Carney, "Dynamic Host Configuration Protocol for IPv6 (DHCPv6)," RFC 3315 (Proposed Standard), Internet Engineering Task Force, Jul. 2003, updated by RFCs 4361, 5494, 6221, 6422, 6644, 7083, 7227, 7283, 7550. [Online]. Available: http://www.ietf.org/rfc/rfc3315.txt

[34] Y. Rekhter, T. Li, and S. Hares, "A Border Gateway Protocol 4 (BGP-4)," RFC 4271 (Draft Standard), Internet Engineering Task Force, Jan. 2006, updated by RFCs 6286, 6608, 6793, 7606, 7607. [Online]. Available: http://www.ietf.org/rfc/rfc4271.txt

[35] J. Postel, "Transmission Control Protocol," RFC 793 (INTERNET STANDARD), Internet Engineering Task Force, Sep. 1981, updated by RFCs 1122, 3168, 6093, 6528. [Online]. Available: http://www.ietf.org/rfc/rfc793.txt

[36] P. Srisuresh and M. Holdrege, "IP Network Address Translator (NAT) Terminology and Considerations," RFC 2663 (Informational), Internet Engineering Task Force, Aug. 1999. [Online]. Available: http://www.ietf.org/rfc/rfc2663.txt

[37] T. H. Ptacek and T. N. Newsham, "Insertion, Evasion, and Denial of Service: Eluding Network Intrusion Detection," Secure Networks, Inc, Tech. Rep., 1 1998.

[38] A. S. Gorton and T. G. Champion, "Combining Evasion Techniques to Avoid Network Intrusion Detection Systems," Skaion Corporation, Tech. Rep., 2004.

[39] CISCO, "Configuring TCP Normalization," http://www.cisco.com/c/en/us/td/docs/security/asa/asa82/configuration/guide/config/conns_tcpnorm.pdf, 2009.

[40] ——, "README.normalize: When operating Snort in inline mode, it is helpful to normalize packets to help minimize the chances of evasion." https://snort.org/faq/readme-normalize, 2015.

[41] C. Stoll, "Stalking the Wily Hacker," *Communications of the ACM (CACM)*, vol. 31, no. 5, pp. 484–497, 1988. [Online]. Available: http://doi.acm.org/10.1145/42411.42412

[42] M. W. Eichin and J. A. Rochlis, "With Microscope and Tweezers: An Analysis of the Internet Virus of November 1988," in *Proceedings of the 1989 IEEE Symposium on Security and Privacy, Oakland, California, USA, May 1-3, 1989*. IEEE Computer Society, 1989, pp. 326–343. [Online]. Available: https://doi.org/10.1109/SECPRI.1989.36307

[43] L. T. Herberlein, G. V. Dias, K. N. Levitt, B. Mukherjee, and J. Wood, "Network attacks and an Ethernet-based network security monitor," in *DOE Security Group Conf*, 1990.

[44] L. T. Herberlein, G. V. Dias, K. N. Levitt, B. Mukherjee, J. Wood, and D. Wolber, "A Network Security Monitor," in *IEEE Symposium on Security and Privacy, Oakland, California, USA*. IEEE Computer Society, 1990, pp. 296–305. [Online]. Available: http://dx.doi.org/10.1109/RISP.1990.63859

[45] S. R. Snapp, S. E. Smaha, D. M. Teal, and T. Grance, "The DIDS (distributed intrusion detection system) prototype," in *USENIX, San Antonio, TX, USA*. USENIX Association, 1992. [Online]. Available: https://www.usenix.org/conference/usenix-summer-1992-technical-conference/dids-distributed-intrusion-detection-system

[46] M. Kenney, "Ping of Death," http://insecure.org/sploits/ping-o-death.html, 1996.

[47] m3lt, "The LAND attack (IP DOS)," http://insecure.org/sploits/land.ip.DOS.html, 1997.

[48] E. Henigin, "Routed broadcast ping DOS attack," http://insecure.org/sploits/routed.broadcast.ping.DOS.html, 1997.

[49] IBM, "UDP_Port_Loopback," https://exchange.xforce.ibmcloud.com/signature/2000202, 2003.

[50] ——, "Snork_Attack," https://exchange.xforce.ibmcloud.com/signature/2000203, 2003.

[51] CAPEC, "CAPEC-304: TCP Null Scan," http://capec.mitre.org/data/definitions/304.html, 2014.

[52] M. Roesch, "Snort: Lightweight Intrusion Detection for Networks," in *13th Conference on Systems Administration (LISA-99), Seattle, WA*, D. W. Parter, Ed. USENIX, 1999, pp. 229–238. [Online]. Available: http://www.usenix.org/publications/library/proceedings/lisa99/roesch.html

[53] S. Schmerl, H. König, U. Flegel, M. Meier, and R. Rietz, "Systematic Signature Engineering by Re-use of Snort Signatures," in *Twenty-Fourth Annual Computer Security Applications Conference, ACSAC 2008, Anaheim, California, USA*, 2008, pp. 23–32. [Online]. Available: http://dx.doi.org/10.1109/ACSAC.2008.20

[54] B. Claise, B. Trammell, and P. Aitken, "Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information," RFC 7011 (INTERNET STANDARD), Internet Engineering Task Force, Sep. 2013. [Online]. Available: http://www.ietf.org/rfc/rfc7011.txt

[55] B. Claise and B. Trammell, "Information Model for IP Flow Information Export (IPFIX)," RFC 7012 (Proposed Standard), Internet Engineering Task Force, Sep. 2013. [Online]. Available: http://www.ietf.org/rfc/rfc7012.txt

[56] Kaspersky, "The Regin Platform: Nation-State Ownage of GSM Networks," Kaspersky, Tech. Rep., November 2014. [Online]. Available: https://cdn.securelist.com/files/2014/11/Kaspersky_Lab_whitepaper_Regin_platform_eng.pdf

[57] Symantec, "Regin: Top-tier espionage tool enables stealthy surveillance," Symantec, Tech. Rep., August 2015. [Online]. Available: http://securityresponse.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/regin-analysis.pdf

[58] OGSYS, *Protocols for X/Open PC Interworking: SMB, Version 2.* The Open Group, 1992. [Online]. Available: https://www2.opengroup.org/ogsys/catalog/c209

[59] T. Dierks and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2," RFC 5246 (Proposed Standard), Internet Engineering Task Force, Aug. 2008, updated by RFCs 5746, 5878, 6176, 7465, 7507, 7568, 7627. [Online]. Available: http://www.ietf.org/rfc/rfc5246.txt

[60] J. Case, K. McCloghrie, M. Rose, and S. Waldbusser, "Introduction to version 2 of the Internet-standard Network Management Framework," RFC 1441 (Historic), Internet Engineering Task Force, Apr. 1993. [Online]. Available: http://www.ietf.org/rfc/rfc1441.txt

[61] J. Case, R. Mundy, D. Partain, and B. Stewart, "Introduction and Applicability Statements for Internet-Standard Management Framework," RFC 3410 (Informational), Internet Engineering Task Force, Dec. 2002. [Online]. Available: http://www.ietf.org/rfc/rfc3410.txt

[62] F. T. Andrews and K. E. Martersteck, "Stored Program Controlled Network: Prologue," *The Bell System Technical Journal*, vol. 61, no. 7, pp. 1575–1577, Sept 1982.

[63] S. Horing, J. Z. Menard, R. E. Staehler, and B. J. Yokelson, "Stored Program Controlled Network: Overview," *The Bell System Technical Journal*, vol. 61, no. 7, pp. 1579–1588, Sept 1982.

[64] J. J. Lawser, R. E. LeCronier, and R. L. Simms, "Stored Program Controlled Network: Generic network plan," *The Bell System Technical Journal*, vol. 61, no. 7, pp. 1589–1598, Sept 1982.

[65] D. L. Tennenhouse, J. M. Smith, W. D. Sincoskie, D. J. Wetherall, and G. J. Minden, "A survey of active network research," *IEEE Communications Magazine*, vol. 35, no. 1, pp. 80–86, Jan 1997.

[66] T. V. Lakshman, T. Nandagopal, R. Ramjee, K. Sabnani, and T. Woo, "The softrouter architecture," in *ACM HOTNETS*, 2004.

[67] L. Yang, R. Dantu, T. Anderson, and R. Gopal, "Forwarding and Control Element Separation (ForCES) Framework," RFC 3746 (Informational), Internet Engineering Task Force, Apr. 2004. [Online]. Available: http://www.ietf.org/rfc/rfc3746.txt

[68] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker, "Ethane: taking control of the enterprise," in *ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, Kyoto, Japan*, J. Murai and K. Cho, Eds. ACM, 2007, pp. 1–12. [Online]. Available: http://doi.acm.org/10.1145/1282380.1282382

[69] N. McKeown, T. Anderson, H. Balakrishnan, G. M. Parulkar, L. L. Peterson, J. Rexford, S. Shenker, and J. S. Turner, "Openflow: enabling innovation in campus networks," *Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008. [Online]. Available: http://doi.acm.org/10.1145/1355734.1355746

[70] I. Ahmad, S. Namal, M. Ylianttila, and A. Gurtov, "Security in software defined networks: A survey," *IEEE Communications Surveys and Tutorials*, vol. 17, no. 4, pp. 2317–2346, 2015. [Online]. Available: http://dx.doi.org/10.1109/COMST.2015.2474118

[71] S. Scott-Hayward, S. Natarajan, and S. Sezer, "A Survey of Security in Software Defined Networks," *IEEE Communications Surveys and Tutorials*, vol. 18, no. 1, pp. 623–654, 2016. [Online]. Available: http://dx.doi.org/10.1109/COMST.2015.2453114

[72] J. François, L. Dolberg, O. Festor, and T. Engel, "Network security through software defined networking: a survey," in *Conference on Principles, Systems and Applications of IP Telecommunications, IPTComm 2014, Chicago, Illinois, USA*. ACM, 2014, pp. 6:1–6:8. [Online]. Available: http://doi.acm.org/10.1145/2670386.2670390

[73] J. Yu and J. Li, "A Parallel NIDS Pattern Matching Engine and Its Implementation on Network Processor," in *International Conference on Security and Management, SAM 2005, Las Vegas, Nevada, USA*, H. R. Arabnia, Ed. CSREA Press, 2005, pp. 375–384.

[74] G. Vasiliadis, S. Antonatos, M. Polychronakis, E. P. Markatos, and S. Ioannidis, "Gnort: High Performance Network Intrusion Detection Using Graphics Processors," in *Recent Advances in Intrusion Detection, 11th International Symposium, RAID 2008, Cambridge, MA, USA*, 2008, pp. 116–134. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-87403-4_7

[75] R. Sommer, V. Paxson, and N. Weaver, "An architecture for exploiting multi-core processors to parallelize network intrusion prevention," *Concurrency and Computation: Practice and Experience*, vol. 21, no. 10, pp. 1255–1279, 2009. [Online]. Available: http://dx.doi.org/10.1002/cpe.1422

[76] R. Smith, C. Estan, and S. Jha, "XFA: Faster Signature Matching with Extended Automata," in *2008 IEEE Symposium on Security and Privacy (S&P 2008), Oakland, California, USA*. IEEE Computer Society, 2008, pp. 187–201. [Online]. Available: http://dx.doi.org/10.1109/SP.2008.14

[77] S. Wu and U. Manber, "A fast algorithm for multi-pattern searching," Tech. Rep., 1994.

[78] G. Vigna and R. A. Kemmerer, "NetSTAT: A Network-Based Intrusion Detection Approach," in *14th Annual Computer Security Applications Conference (ACSAC 1998), Scottsdale, AZ, USA*. IEEE Computer Society, 1998, pp. 25–34. [Online]. Available: http://dx.doi.org/10.1109/CSAC.1998.738566

[79] C. Krügel, T. Toth, and E. Kirda, "SPARTA, a Mobile Agent Based Instrusion Detection System," in *Advances in Network and Distributed Systems Security, IFIP TC11 WG11.4 First Annual Working Conference on Network Security, Leuven, Belgium*, ser. IFIP Conference Proceedings, B. D. Decker, F. Piessens, J. Smits, and E. V. Herreweghen, Eds., vol. 206. Kluwer, 2001, pp. 187–198. [Online]. Available: http://dx.doi.org/10.1007/0-306-46958-8_13

[80] G. Ramachandran and D. Hart, "A P2P intrusion detection system based on mobile agents," in *42nd Annual Southeast Regional Conference, 2004, Huntsville, Alabama, USA*, S. Yoo and L. H. Etzkorn, Eds. ACM, 2004, pp. 185–190. [Online]. Available: http://doi.acm.org/10.1145/986537.986581

[81] R. Janakiraman, M. Waldvogel, and Q. Zhang, "Indra: A peer-to-peer approach to network intrusion detection and prevention," in *12th IEEE International Workshops on Enabling Technologies (WETICE 2003), Infrastructure for Collaborative Enterprises, Linz, Austria*. IEEE Computer Society, 2003, pp. 226–231. [Online]. Available: http://dx.doi.org/10.1109/ENABL.2003.1231412

[82] D. Ye, Q. Bai, M. Zhang, and Z. Ye, "P2P distributed intrusion detections by using mobile agents," in *7th IEEE/ACIS International Conference on Computer and Information Science, IEEE/ACIS ICIS 2008, Portland, Oregon, USA*, R. Y. Lee, Ed. IEEE Computer Society, 2008, pp. 259–265. [Online]. Available: http://dx.doi.org/10.1109/ICIS.2008.21

[83] S. T. Zargar, H. Takabi, and J. B. D. Joshi, "DCDIDP: A distributed, collaborative, and data-driven intrusion detection and prevention framework for cloud computing environments," in *7th International Conference on Collaborative Computing: Networking, Applications and Worksharing, CollaborateCom*

2011, Orlando, FL, USA, D. Georgakopoulos and J. B. D. Joshi, Eds. ICST / IEEE, 2011, pp. 332–341. [Online]. Available: http://dx.doi.org/10.4108/icst.collaboratecom.2011.247158

[84] M. V. Gundy and H. Chen, "Noncespaces: Using Randomization to Enforce Information Flow Tracking and Thwart Cross-Site Scripting Attacks," in *Network and Distributed System Security Symposium, NDSS 2009, San Diego, California, USA*, 2009. [Online]. Available: http://www.isoc.org/isoc/conferences/ndss/09/pdf/03.pdf

[85] P. Würzinger, C. Platzer, C. Ludl, E. Kirda, and C. Kruegel, "SWAP: mitigating XSS attacks using a reverse proxy," in *ICSE Workshop on Software Engineering for Secure Systems, SESS 2009, Vancouver, BC, Canada*. IEEE Computer Society, 2009, pp. 33–39. [Online]. Available: http://dx.doi.org/10.1109/IWSESS.2009.5068456

[86] H. Shahriar and M. Zulkernine, "S2XS2: A Server Side Approach to Automatically Detect XSS Attacks," in *IEEE Ninth International Conference on Dependable, Autonomic and Secure Computing, DASC 2011, Sydney, Australia*. IEEE Computer Society, 2011, pp. 7–14. [Online]. Available: http://dx.doi.org/10.1109/DASC.2011.26

[87] T. Jim, N. Swamy, and M. Hicks, "Defeating script injection attacks with browser-enforced embedded policies," in *16th International Conference on World Wide Web, WWW 2007, Banff, Alberta, Canada*, C. L. Williamson, M. E. Zurko, P. F. Patel-Schneider, and P. J. Shenoy, Eds. ACM, 2007, pp. 601–610. [Online]. Available: http://doi.acm.org/10.1145/1242572.1242654

[88] M. T. Louw and V. N. Venkatakrishnan, "Blueprint: Robust Prevention of Cross-site Scripting Attacks for Existing Browsers," in *30th IEEE Symposium on Security and Privacy (S&P 2009), Oakland, California, USA*. IEEE Computer Society, 2009, pp. 331–346. [Online]. Available: http://dx.doi.org/10.1109/SP.2009.33

[89] O. Hallaraker and G. Vigna, "Detecting Malicious JavaScript Code in Mozilla," in *10th International Conference on Engineering of Complex Computer Systems (ICECCS 2005), Shanghai, China*. IEEE Computer Society, 2005, pp. 85–94. [Online]. Available: http://dx.doi.org/10.1109/ICECCS.2005.35

[90] P. Vogt, F. Nentwich, N. Jovanovic, E. Kirda, C. Krügel, and G. Vigna, "Cross Site Scripting Prevention with Dynamic Data Tainting and Static Analysis," in *Network and Distributed System Security Symposium, NDSS 2007, San Diego, California, USA*. The Internet Society, 2007. [Online]. Available: http://www.isoc.org/isoc/conferences/ndss/07/papers/cross-site-scripting_prevention.pdf

[91] B. Stock, S. Lekies, T. Mueller, P. Spiegel, and M. Johns, "Precise Client-side Protection against DOM-based Cross-Site Scripting," in *23rd USENIX Security Symposium, San Diego, CA, USA*, K. Fu and J. Jung, Eds. USENIX Association, 2014, pp. 655–670. [Online]. Available: https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/stock

[92] P. Raman, *JaSPIn: JavaScript Based Anomaly Detection of Cross-site Scripting Attacks*, ser. Master thesis. Carleton University (Canada), 2008.

[93] P. D. Ryck, L. Desmet, W. Joosen, and F. Piessens, "Automatic and Precise Client-Side Protection against CSRF Attacks," in *Computer Security - ESORICS 2011 - 16th European Symposium on Research in Computer Security, Leuven, Belgium*, ser. Lecture Notes in Computer Science, V. Atluri and C. Díaz, Eds., vol. 6879. Springer, 2011, pp. 100–116. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-23822-2_6

[94] C. Curtsinger, B. Livshits, B. G. Zorn, and C. Seifert, "ZOZZLE: Fast and Precise In-Browser JavaScript Malware Detection," in *20th USENIX Security Symposium, San Francisco, CA, USA*. USENIX Association, 2011. [Online]. Available: http://static.usenix.org/events/sec11/tech/full_papers/Curtsinger.pdf

[95] B. Stritter, F. C. Freiling, H. König, R. Rietz, S. Ullrich, A. von Gernler, F. Erlacher, and F. Dressler, "Cleaning up Web 2.0's Security Mess-at Least Partly," *IEEE Security & Privacy*, vol. 14, no. 2, pp. 48–57, 2016. [Online]. Available: http://dx.doi.org/10.1109/MSP.2016.31

[96] M. Casado, "Architectural support for security management in enterprise networks," Ph.D. dissertation, Stanford, CA, USA, 2007.

[97] S. Alexander and R. Droms, "DHCP Options and BOOTP Vendor Extensions," RFC 1533 (Proposed Standard), Internet Engineering Task Force, Oct. 1993, obsoleted by RFC 2132. [Online]. Available: http://www.ietf.org/rfc/rfc1533.txt

[98] A. Conta, S. Deering, and M. Gupta, "Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification," RFC 4443 (Draft Standard), Internet Engineering Task Force, Mar. 2006, updated by RFC 4884. [Online]. Available: http://www.ietf.org/rfc/rfc4443.txt

[99] R. Vida and L. Costa, "Multicast Listener Discovery Version 2 (MLDv2) for IPv6," RFC 3810 (Proposed Standard), Internet Engineering Task Force, Jun. 2004, updated by RFC 4604. [Online]. Available: http://www.ietf.org/rfc/rfc3810.txt

[100] T. Narten, E. Nordmark, W. Simpson, and H. Soliman, "Neighbor Discovery for IP version 6 (IPv6)," RFC 4861 (Draft Standard), Internet Engineering

Task Force, Sep. 2007, updated by RFCs 5942, 6980, 7048, 7527, 7559. [Online]. Available: http://www.ietf.org/rfc/rfc4861.txt

[101] T. Chown and S. Venaas, "Rogue IPv6 Router Advertisement Problem Statement," RFC 6104 (Informational), Internet Engineering Task Force, Feb. 2011. [Online]. Available: http://www.ietf.org/rfc/rfc6104.txt

[102] E. Nordmark, "Stateless IP/ICMP Translation Algorithm (SIIT)," RFC 2765 (Proposed Standard), Internet Engineering Task Force, Feb. 2000, obsoleted by RFC 6145. [Online]. Available: http://www.ietf.org/rfc/rfc2765.txt

[103] G. Tsirtsis and P. Srisuresh, "Network Address Translation - Protocol Translation (NAT-PT)," RFC 2766 (Historic), Internet Engineering Task Force, Feb. 2000, obsoleted by RFC 4966, updated by RFC 3152. [Online]. Available: http://www.ietf.org/rfc/rfc2766.txt

[104] J. C. Mogul, R. F. Rashid, and M. J. Accetta, "The Packet Filter: An Efficient Mechanism for User-level Network Code," in *Eleventh ACM Symposium on Operating System Principles, SOSP 1987, Austin, Texas, USA*, L. Belady, Ed. ACM, 1987, pp. 39–51. [Online]. Available: http://doi.acm.org/10.1145/41457.37505

[105] S. Ioannidis, A. D. Keromytis, S. M. Bellovin, and J. M. Smith, "Implementing a distributed firewall," in *CCS 2000, Proceedings of the 7th ACM Conference on Computer and Communications Security, Athens, Greece*, D. Gritzalis, S. Jajodia, and P. Samarati, Eds. ACM, 2000, pp. 190–199. [Online]. Available: http://doi.acm.org/10.1145/352600.353052

[106] CISCO, "VLAN Security White Paper," http://web.archive.org/web/20141123062129/http://www.cisco.com/en/US/products/hw/switches/ps708/products_white_paper09186a008013159f.shtml, 2014.

[107] M. Casado, T. Garfinkel, A. Akella, M. J. Freedman, D. Boneh, and N. McKeown, "SANE: A Protection Architecture for Enterprise Networks," in *15th USENIX Security Symposium, Vancouver, BC, Canada*, A. D. Keromytis, Ed. USENIX Association, 2006. [Online]. Available: https://www.usenix.org/conference/15th-usenix-security-symposium/sane-protection-architecture-enterprise-networks

[108] S. A. Mehdi, J. Khalid, and S. A. Khayam, "Revisiting Traffic Anomaly Detection Using Software Defined Networking," in *Recent Advances in Intrusion Detection - 14th International Symposium, RAID 2011, Menlo Park, CA, USA*, 2011, pp. 161–180. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-23644-0_9

[109] Y. Zhang, "An adaptive flow counting method for anomaly detection in SDN," in *Conference on emerging Networking Experiments and Technologies, CoNEXT '13, Santa Barbara, CA, USA*, K. C. Almeroth, L. Mathy, K. Papagiannaki, and V. Misra, Eds. ACM, 2013, pp. 25–30. [Online]. Available: http://doi.acm.org/10.1145/2535372.2535411

[110] S. Shin, P. A. Porras, V. Yegneswaran, M. W. Fong, G. Gu, and M. Tyson, "FRESCO: Modular Composable Security Services for Software-Defined Networks," in *20th Annual Network and Distributed System Security Symposium, NDSS 2013, San Diego, California, USA*. The Internet Society, 2013. [Online]. Available: http://www.internetsociety.org/events/ndss-symposium-2013

[111] A. Zaalouk, R. Khondoker, R. Marx, and K. M. Bayarou, "OrchSec: An orchestrator-based architecture for enhancing network-security using Network Monitoring and SDN Control functions," in *2014 IEEE Network Operations and Management Symposium, NOMS 2014, Krakow, Poland*. IEEE, 2014, pp. 1–9. [Online]. Available: http://dx.doi.org/10.1109/NOMS.2014.6838409

[112] A. Srivastava and J. T. Giffin, "Tamper-Resistant, Application-Aware Blocking of Malicious Network Connections," in *Recent Advances in Intrusion Detection, 11th International Symposium, RAID 2008, Cambridge, MA, USA*, 2008, pp. 39–58. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-87403-4_3

[113] A. Brinner and R. Rietz, "Verbesserung der Netzsicherheit in virtualisierten Umgebungen mit Hilfe von OpenFlow," in *Sicherheit 2014: Sicherheit, Schutz und Zuverlässigkeit, Beiträge der 7. Jahrestagung des Fachbereichs Sicherheit der Gesellschaft für Informatik e.V. (GI), Wien, Österreich*, ser. LNI, S. Katzenbeisser, V. Lotz, and E. R. Weippl, Eds., vol. 228. GI, 2014, pp. 79–89. [Online]. Available: http://subs.emis.de/LNI/Proceedings/Proceedings228/article27.html

[114] B. Pfaff, J. Pettit, K. Amidon, M. Casado, T. Koponen, and S. Shenker, "Extending Networking into the Virtualization Layer," in *Eight ACM Workshop on Hot Topics in Networks (HotNets-VIII), HOTNETS '09, New York City, NY, USA*, L. Subramanian, W. E. Leland, and R. Mahajan, Eds. ACM SIGCOMM, 2009. [Online]. Available: http://conferences.sigcomm.org/hotnets/2009/papers/hotnets2009-final143.pdf

[115] B. Pfaff, J. Pettit, T. Koponen, E. J. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar, K. Amidon, and M. Casado, "The Design and Implementation of Open vSwitch," in *12th USENIX Symposium on Networked Systems Design and Implementation, NSDI 15, Oakland, CA, USA*. USENIX Association, 2015, pp. 117–130. [Online]. Available: https://www.usenix.org/conference/nsdi15/technical-sessions/presentation/pfaff

[116] C. Finseth, "An Access Control Protocol, Sometimes Called TACACS," RFC 1492 (Informational), Internet Engineering Task Force, Jul. 1993. [Online]. Available: http://www.ietf.org/rfc/rfc1492.txt

[117] B. Aboba, L. Blunk, J. Vollbrecht, J. Carlson, and H. Levkowetz, "Extensible Authentication Protocol (EAP)," RFC 3748 (Proposed Standard), Internet Engineering Task Force, Jun. 2004, updated by RFCs 5247, 7057. [Online]. Available: http://www.ietf.org/rfc/rfc3748.txt

[118] C. Rigney, S. Willens, A. Rubens, and W. Simpson, "Remote Authentication Dial In User Service (RADIUS)," RFC 2865 (Draft Standard), Internet Engineering Task Force, Jun. 2000, updated by RFCs 2868, 3575, 5080, 6929. [Online]. Available: http://www.ietf.org/rfc/rfc2865.txt

[119] K. Zeilenga, "Lightweight Directory Access Protocol (LDAP): Technical Specification Road Map," RFC 4510 (Proposed Standard), Internet Engineering Task Force, Jun. 2006. [Online]. Available: http://www.ietf.org/rfc/rfc4510.txt

[120] R. Hinden and S. Deering, "IP Version 6 Addressing Architecture," RFC 4291 (Draft Standard), Internet Engineering Task Force, Feb. 2006, updated by RFCs 5952, 6052, 7136, 7346, 7371. [Online]. Available: http://www.ietf.org/rfc/rfc4291.txt

[121] S. Deering, W. Fenner, and B. Haberman, "Multicast Listener Discovery (MLD) for IPv6," RFC 2710 (Proposed Standard), Internet Engineering Task Force, Oct. 1999, updated by RFCs 3590, 3810. [Online]. Available: http://www.ietf.org/rfc/rfc2710.txt

[122] J. Mogul, "Broadcasting Internet datagrams in the presence of subnets," RFC 922 (INTERNET STANDARD), Internet Engineering Task Force, Oct. 1984. [Online]. Available: http://www.ietf.org/rfc/rfc922.txt

[123] D. Johnson, C. Perkins, and J. Arkko, "Mobility Support in IPv6," RFC 3775 (Proposed Standard), Internet Engineering Task Force, Jun. 2004, obsoleted by RFC 6275. [Online]. Available: http://www.ietf.org/rfc/rfc3775.txt

[124] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: rapid prototyping for software-defined networks," in *9th ACM Workshop on Hot Topics in Networks. HotNets 2010, Monterey, CA, USA*, G. G. Xie, R. Beverly, R. Morris, and B. Davie, Eds. ACM, 2010, p. 19. [Online]. Available: http://doi.acm.org/10.1145/1868447.1868466

[125] K. Xinidis, I. Charitakis, S. Antonatos, K. G. Anagnostakis, and E. P. Markatos, "An Active Splitter Architecture for Intrusion Detection and Prevention," *IEEE Transaction on Dependable and Secure Computing*, vol. 3, no. 1, pp. 31–44, 2006.

[126] M. Vallentin, R. Sommer, J. Lee, C. Leres, V. Paxson, and B. Tierney, "The NIDS Cluster: Scalable, Stateful Network Intrusion Detection on Commodity Hardware," in *RAID*, ser. Lecture Notes in Computer Science, C. Krügel, R. Lippmann, and A. Clark, Eds., vol. 4637. Springer, 2007, pp. 107–126.

[127] R. Smith, N. Goyal, J. Ormont, K. Sankaralingam, and C. Estan, "Evaluating GPUs for network packet signature matching," in *ISPASS*. IEEE, 2009, pp. 175–184.

[128] M. A. Jamshed, J. Lee, S. Moon, I. Yun, D. Kim, S. Lee, Y. Yi, and K. Park, "Kargus: a highly-scalable software-based intrusion detection system," in *ACM Conference on Computer and Communications Security*, T. Yu, G. Danezis, and V. D. Gligor, Eds. ACM, 2012, pp. 317–328.

[129] B. M. Rogers, A. Krishna, G. B. Bell, K. V. Vu, X. Jiang, and Y. Solihin, "Scaling the bandwidth wall: challenges in and avenues for cmp scaling," in *ISCA*, S. W. Keckler and L. A. Barroso, Eds. ACM, 2009, pp. 371–382.

[130] R. Rietz, M. Vogel, F. Schuster, and H. König, "Parallelization of network intrusion detection systems under attack conditions," in *Detection of Intrusions and Malware, and Vulnerability Assessment - 11th International Conference, DIMVA, Egham, UK*, ser. Lecture Notes in Computer Science, S. Dietrich, Ed., vol. 8550. Springer, 2014, pp. 172–191. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-08509-8_10

[131] F. Massicotte, F. Gagnon, Y. Labiche, L. C. Briand, and M. Couture, "Automatic evaluation of intrusion detection systems," in *ACSAC*. IEEE Computer Society, 2006, pp. 361–370.

[132] G. M. Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities," ser. AFIPS '67 (Spring). New York, NY, USA: ACM, 1967, pp. 483–485. [Online]. Available: http://doi.acm.org/10.1145/1465482.1465560

[133] S. Eyerman and L. Eeckhout, "Modeling critical sections in Amdahl's law and its implications for multicore design," in *ISCA*, A. Seznec, U. C. Weiser, and R. Ronen, Eds. ACM, 2010, pp. 362–370.

[134] F. Fusco and L. Deri, "High speed network traffic analysis with commodity multicore systems," in *Internet Measurement Conference*, M. Allman, Ed. ACM, 2010, pp. 218–224.

[135] G. Vasiliadis, M. Polychronakis, and S. Ioannidis, "MIDeA: a multi-parallel intrusion detection architecture," in *ACM Conference on Computer and Communications Security*, Y. Chen, G. Danezis, and V. Shmatikov, Eds. ACM, 2011, pp. 297–308.

[136] L. Yang, R. Karim, V. Ganapathy, and R. Smith, "Improving NFA-Based Signature Matching Using Ordered Binary Decision Diagrams," in *RAID*, ser. LNCS, S. Jha, R. Sommer, and C. Kreibich, Eds., vol. 6307. Springer, 2010, pp. 58–78.

[137] A. V. Aho and M. J. Corasick, "Efficient String Matching: An Aid to Bibliographic Search," *Communications of the ACM (CACM)*, vol. 18, no. 6, pp. 333–340, 1975. [Online]. Available: http://doi.acm.org/10.1145/360825.360855

[138] T. Limmer and F. Dressler, "Dialog-based payload aggregation for intrusion detection," in *17th ACM Conference on Computer and Communications Security, CCS 2010, Chicago, Illinois, USA*, E. Al-Shaer, A. D. Keromytis, and V. Shmatikov, Eds. ACM, 2010, pp. 708–710. [Online]. Available: http://doi.acm.org/10.1145/1866307.1866405

[139] J. Amann and R. Sommer, "Providing Dynamic Control to Passive Network Security Monitoring," in *Research in Attacks, Intrusions, and Defenses - 18th International Symposium, RAID 2015, Kyoto, Japan*, ser. Lecture Notes in Computer Science, H. Bos, F. Monrose, and G. Blanc, Eds., vol. 9404. Springer, 2015, pp. 133–152. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-26362-5_7

[140] O. Ismail, M. Etoh, Y. Kadobayashi, and S. Yamaguchi, "A Proposal and Implementation of Automatic Detection/Collection System for Cross-Site Scripting Vulnerability," in *18th International Conference on Advanced Information Networking and Applications (AINA 2004), Fukuoka, Japan*. IEEE Computer Society, 2004, pp. 145–151. [Online]. Available: http://dx.doi.org/10.1109/AINA.2004.1283902

[141] M. Heiderich, T. Frosch, and T. Holz, "IceShield: Detection and Mitigation of Malicious Websites with a Frozen DOM," in *Recent Advances in Intrusion Detection - 14th International Symposium, RAID 2011, Menlo Park, CA, USA*, 2011, pp. 281–300. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-23644-0_15

[142] R. Wang, X. Jia, Q. Li, and S. Zhang, "Machine Learning Based Cross-Site Scripting Detection in Online Social Network," in *2014 IEEE International Conference on High Performance Computing and Communications, 6th IEEE International Symposium on Cyberspace Safety and Security, 11th IEEE International Conference on Embedded Software and Systems, HPCC/CSS/ICESS 2014, Paris, France*. IEEE, 2014, pp. 823–826. [Online]. Available: http://dx.doi.org/10.1109/HPCC.2014.137

[143] P. H. Phung, D. Sands, and A. Chudnov, "Lightweight self-protecting JavaScript," in *Proceedings of the 2009 ACM Symposium on Information, Computer and Communications Security, ASIACCS 2009, Sydney, Australia*, W. Li, W. Susilo, U. K. Tupakula, R. Safavi-Naini, and V. Varadharajan,

Eds. ACM, 2009, pp. 47–60. [Online]. Available: http://doi.acm.org/10.1145/1533057.1533067

[144] Y. Nadji, P. Saxena, and D. Song, "Document Structure Integrity: A Robust Basis for Cross-site Scripting Defense," in *Network and Distributed System Security Symposium, NDSS 2009, San Diego, California, USA*, 2009. [Online]. Available: http://www.isoc.org/isoc/conferences/ndss/09/pdf/01.pdf

[145] S. Tang, C. Grier, O. Aciiçmez, and S. T. King, "Alhambra: a system for creating, enforcing, and testing browser security policies," in *Proceedings of the 19th International Conference on World Wide Web, WWW 2010, Raleigh, North Carolina, USA*, M. Rappa, P. Jones, J. Freire, and S. Chakrabarti, Eds. ACM, 2010, pp. 941–950. [Online]. Available: http://doi.acm.org/10.1145/1772690.1772786

[146] L. A. Meyerovich and V. B. Livshits, "ConScript: Specifying and Enforcing Fine-Grained Security Policies for JavaScript in the Browser," in *31st IEEE Symposium on Security and Privacy, S&P 2010, Berkeley/Oakland, California, USA*. IEEE Computer Society, 2010, pp. 481–496. [Online]. Available: http://dx.doi.org/10.1109/SP.2010.36

[147] E. Kirda, C. Krügel, G. Vigna, and N. Jovanovic, "Noxes: a client-side solution for mitigating cross-site scripting attacks," in *2006 ACM Symposium on Applied Computing (SAC), Dijon, France*, H. Haddad, Ed. ACM, 2006, pp. 330–337. [Online]. Available: http://doi.acm.org/10.1145/1141277.1141357

[148] M. Johns, B. Engelmann, and J. Posegga, "XSSDS: Server-Side Detection of Cross-Site Scripting Attacks," in *Twenty-Fourth Annual Computer Security Applications Conference, ACSAC 2008, Anaheim, California, USA*, 2008, pp. 335–344. [Online]. Available: http://dx.doi.org/10.1109/ACSAC.2008.36

[149] P. Bisht and V. N. Venkatakrishnan, "XSS-GUARD: Precise Dynamic Prevention of Cross-Site Scripting Attacks," in *Detection of Intrusions and Malware, and Vulnerability Assessment, 5th International Conference, DIMVA Paris, France*, ser. Lecture Notes in Computer Science, D. Zamboni, Ed., vol. 5137. Springer, 2008, pp. 23–43. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-70542-0_2

[150] A. M. Turing, "On Computable Numbers, with an Application to the Entscheidungsproblem," *Proceedings of the London Mathematical Society*, vol. s2-42, no. 1, pp. 230–265, 1937. [Online]. Available: http://plms.oxfordjournals.org/content/s2-42/1/230.short

[151] J. Elson and A. Cerpa, "Internet Content Adaptation Protocol (ICAP)," RFC 3507 (Informational), Internet Engineering Task Force, Apr. 2003. [Online]. Available: http://www.ietf.org/rfc/rfc3507.txt

[152] C. Cortes and V. Vapnik, "Support-vector networks," *Machine Learning*, vol. 20, no. 3, pp. 273–297, 1995. [Online]. Available: http://dx.doi.org/10.1007/BF00994018