



SECURITY CONTROLS - SECURITY INFORMATION AND EVENT MONITORING (SIEM)

Prof. Dr. Bernhard Tellenbach

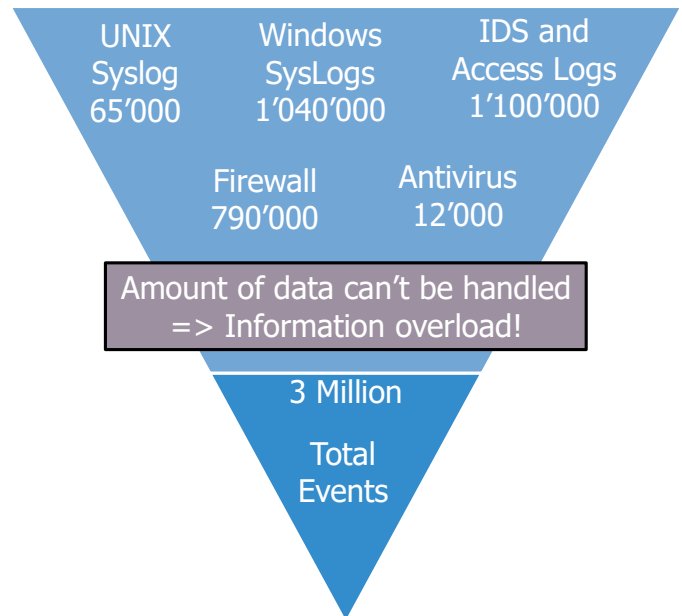
- Security controls revisited
 - Detecting a needle in a stack of needles
- Log Management
 - Basic architecture
- SIEM - Security Information and Event Management
 - Basic architecture
 - Event correlation
 - In AlienVault USM SIEM
 - Use cases
 - Threat Feeds
- The Pyramid of Pain (indicators)

- You can explain **what a SIEM system does** and list and explain important components (sensors like **NIDS** and **HIDS**, log collection and normalization)
- You know why **the correlation engine** is an **important building block** of a SIEM and you can explain simple and more complex **examples of correlation “rules”**
- You know why an **asset inventory** and a **vulnerability scanner** are important components of a SIEM and you can explain **their role in prioritizing alerts**
- You know the concept of **the Pyramid of Pain** and you can discuss the **pros and cons of different indicators** (of compromise)

- Security controls are **safeguards** or countermeasures to **avoid**, **detect**, **counteract**, or **minimize security risks** to physical property, information, computer systems, or other assets.
- CIS Critical Security Controls:
 - Many controls and many building blocks to implement the controls
 - Firewalls, antivirus, IDS, vulnerability scanners, software inventories, ...
- A lot of data from (the building blocks of) the controls:
 - Preventive controls: Data on **a potential attack that failed** (e.g., login)
 - Detective controls: Data on **a potential attack just happening** (e.g., IDS)
 - Corrective controls: Data on **corrective actions** (e.g., DDoS offloading)
- How to extract the relevant information?
=> Find THE needle in a stack of needles



- Components produce (log-)data
 - Mid-sized company: **millions of events/day**
 - Large company: **billions of events/day**
 - Logging network flow data
 - SWITCH network: ~50-200 mio flows/h
- First challenge: **Log management**
- Second challenge: **Making sense of security-relevant log-data**
- Both challenges addressed with a **Security Information and Event Management System (SIEM)**
 - ...or separate log management and SIEM



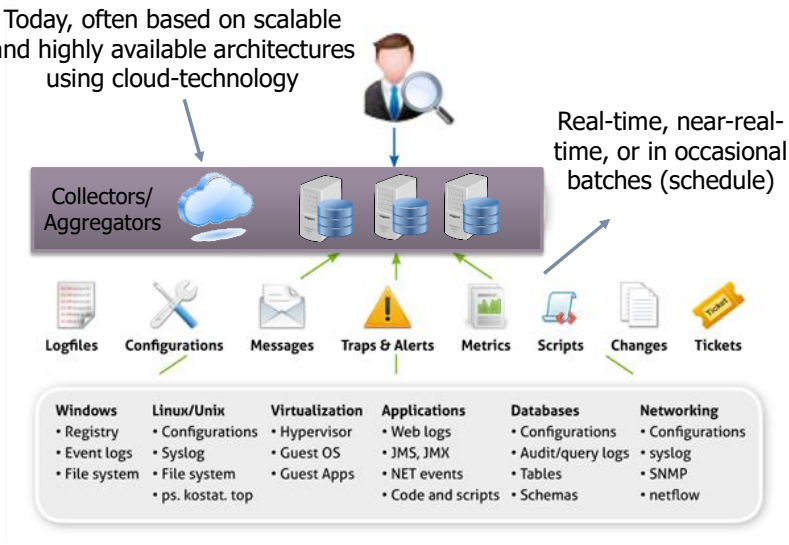
- Before you can make sense of logs:
 - You need to **have logs!**
 - You need to **solidify the logging** process
 - What is logged where and why?
 - How long are logs stored and where?
 - Consider **legal aspects** of logging
 - Depends on data in the logs, i.e., behavioural tracking can be problematic
- **Log management solutions** help with:
 - Collection
 - Retention
 - Indexing/Parsing and searching
 - Reporting
 - ... for all types of log data



There are several documents that offer guidance with regard to the legal aspect of logging. One example is the guide on Internet and e-mail surveillance for the private sector:

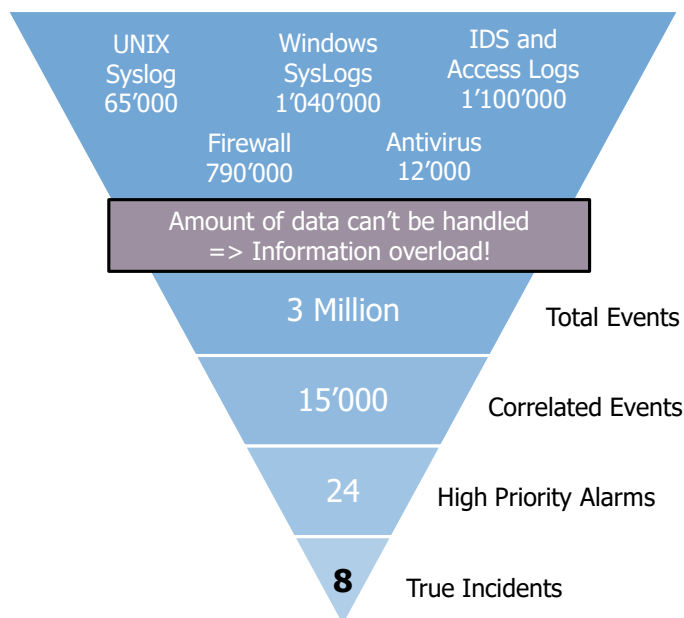
<http://www.edoeb.admin.ch/datenschutz/00763/00983/00988/index.html>

Today, often based on scalable and highly available architectures using cloud-technology

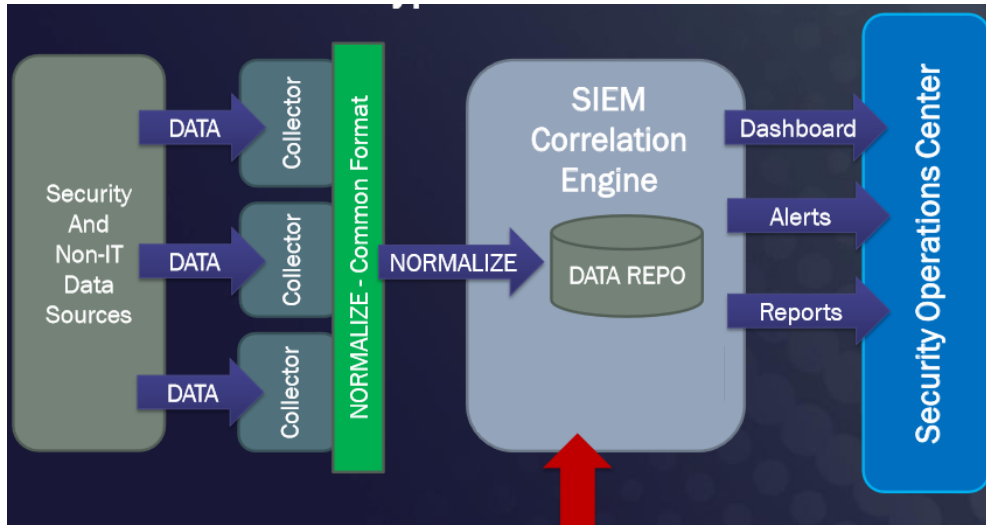


3. Log Monitoring: Dashboards to monitor and review log data and the results of automated analysis
2. Log Analysis + Storage: One or multiple log servers that receive log data or copy log data from the hosts in the first tier
1. Log Generation: The hosts/services/apps that generate the log data

- We need more than log management
- One event/log line is rarely enough to identify an attack reliably
- We need to **correlate** events
- We need to **prioritize** events



- SIEM System – Security Information and Event Management System
- Gathering, aggregating, correlating, analysing, prioritizing and presenting information from disparate systems
- A SIEM correlates events (the “E” in SIEM) and raise alerts and prioritize them
- Usage Scenarios
 - Security Operations Center (SOC) – Analysts 24/7, chase alerts
 - Mini-SOC / “morning after” – Analysts 1/24, review and drill down
 - “Automated SOC” / alert + investigate – Configure & forget, alerts only
 - Compliance status reporting - Review reports/views weekly/monthly
- Stealthy attacks and a shift to advanced tactics and techniques requires SIEM to focus on what is normal (Baseline) and zero in on the exceptions (“Anomalies”)



- Many commercial systems:
 - IBM QRadar, HP ArcSight, Splunk, McAfee Enterprise Security Manager,...
- Notable open-source system:
 - AlienVault's Open Source SIEM (OSSIM)
 - Wazuh
 - SIEMonster
 - ...

- A few of the **key components** that are part of **most SIEM installations**
 - **Asset directory** - Asset value for **risk calculation/prioritization** and more
 - **Vulnerability scanner** - Correlate alerts for a host with information about its vulnerabilities
- Main **sources** of log events:
 - Firewall logs, authentication logs, service- and system logs
 - Blocked packets, failed login attempts, access violations (files, memory, ...),...
 - Host, network and service availability monitoring
 - Unusual load, availability problems
 - Antivirus system
 - Detection of a malicious file, suspicious/malicious behaviour of a process,...
 - Honeypots
 - **TODO**
 - **Host- & Network-based Intrusion Detection System (HIDS/NIDS)**

Another important source is **Data Leakage/Loss Prevention (DLP) Systems**.

A DLP system detects potential data breaches/data ex-filtration transmissions and prevents them by monitoring, detecting and blocking sensitive data while *in use* (endpoint actions), *in motion* (network traffic), and *at rest* (data storage). The terms “data loss” and “data leakage” are related and are often used interchangeably. Data loss incidents turn into data leak incidents in cases where media containing sensitive information is lost and subsequently acquired by an unauthorized party. However, a data leak is possible without losing the data on the originating side.

Source: https://en.wikipedia.org/wiki/Data_loss_prevention_software

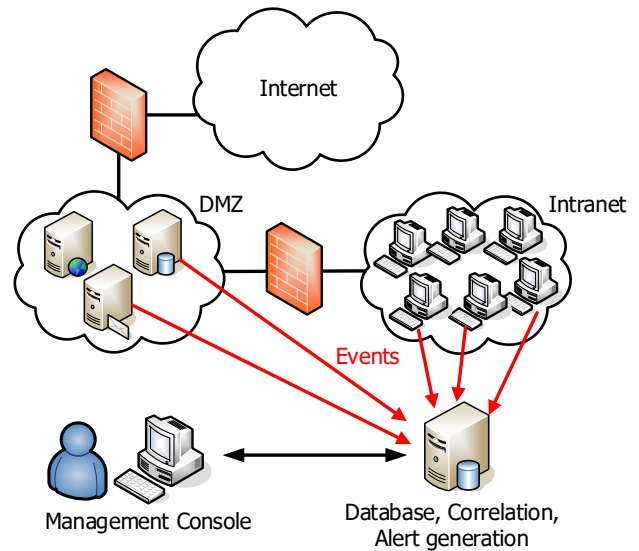
Examples of sources:

- Firewall logs, authentication logs, service- and system logs
 - E.g., Windows event log, linux syslog, webserver logs, ssh daemon logs, ...
- Host- and Network based Intrusion Detection System (HIDS/NIDS)
 - E.g., Snort, Bro or Suricata
- Host, network and service availability monitoring
 - E.g., Nagios, Zabbix
- Data Leakage/Loss Prevention (DLP) System
 - E.g., Symantec DLP, SecureTrust DLP, McAfee Total Protection for DLP
- Antivirus system
 - E.g., Sophos, McAfee, F-Secure, Bitdefender, ...
- Honeypots
 - E.g., honeyd, wifi-honey, kippo, KFSensor, SecuriOT

Intrusion Detection Systems

- Intrusion Detection Systems (IDS) **monitor network** and/or **system activity** to **detect attacks**, attack attempts and **general abuse**
- Intrusion Prevention Systems (IPS) **add a component** to **prevent/mitigate** detected attacks, attack attempts and general abuse.
 - A **false positive** might directly **affect your business** (e.g., preventing a potential customer from accessing your web-shop)
 - There may **already** have been **some damage done** by the time the intrusion is detected
- There are two basic types: **Host-based** systems (HIDS/HIPS) and **network-based** systems (NIDS/NIPS)
- In general, they consist of multiple components: Host or network sensors, centralised database, correlation engine, ,management console

- Monitor individual hosts - **sensors** (software) are installed on the hosts that are monitored
- Things such sensors might look at:
 - System calls
 - Application logs
 - File modifications
 - Processes running on the host
 - Network data from/to this host
- Examples: OSSEC, AIDE, Samhain, Fail2Ban, Sagan



System Call: System calls provide an essential interface between a process and the operating system

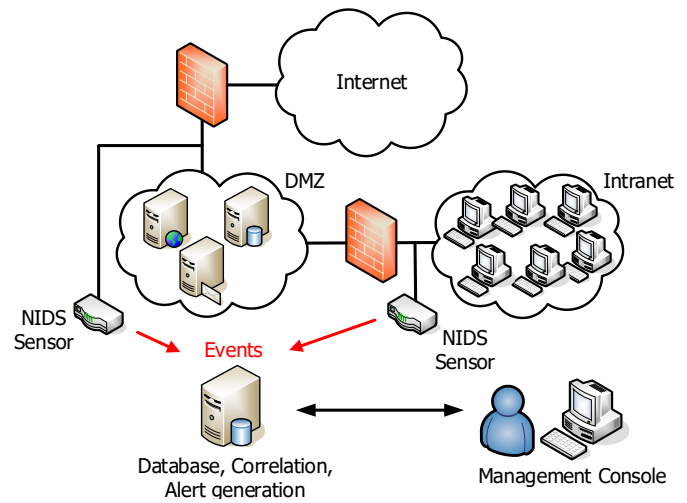
The architecture of most modern processors, with the exception of some embedded systems, involves a security model. For example, the rings model specifies multiple privilege levels under which software may be executed: a program is usually limited to its own address space so that it cannot access or modify other running programs or the operating system itself, and is usually prevented from directly manipulating hardware devices (e.g. the frame buffer or network devices).

However, many normal applications obviously need access to these components, so system calls are made available by the operating system to provide well-defined, safe implementations for such operations. The operating system executes at the highest level of privilege, and allows applications to request services via system calls, which are often initiated via interrupts. An interrupt automatically puts the CPU into some elevated privilege level, and then passes control to the kernel, which determines whether the calling program should be granted the requested service. If the service is granted, the kernel executes a specific set of instructions over which the calling program has no direct control, returns the privilege level to that of the calling program, and then returns control to the calling program.

Source: https://en.wikipedia.org/wiki/System_call

- The **actions** that HIPS use to shut down a threat depends on its **focus/purpose**
 - Quarantine problematic files/executables
 - Restoring files from storage
 - Suspending user accounts
 - Updating firewall settings (e.g., blocking an IP address)
 - Starting or killing processes
 - Shutting down systems
- An example of a simple and very focused HIPS is *Fail2ban*
 - Scans log files (e.g., /var/log/apache/error_log) and detects IPs that show malicious behavior such as too many unsuccessful login attempts or probing for vulnerabilities.
 - Prevention is done by updating firewall rules to reject the IP addresses for a specified amount of time

- Monitor **network segments** - sensors are **devices in the network** that see the desired traffic
 - E.g., attached to the monitor port of a switch, a network tap or as part of a security gateway
- Analyses **network traffic** for malicious (or suspicious) activity such as known attacks, network recon, ...
- Examples: Snort, Bro, Suricata, Sagan




```
alert tcp $EXTERNAL_NET any -> 10.200.0.0/24 80 (msg:"WEB-IIS
CodeRed v2 root.exe access"; flow:to_server,established;
uricontent:"/root.exe"; nocase; classtype:web application-attack;
reference:url,www.cert.org/advisories/CA-2001_19.html; sid:1255;
rev:7;)
```

- **\$EXTERNAL_NET**: a variable in Suricata matching all IPs not in \$HOME_NET
- **\$HOME_NET** is defined as any IP within a user-defined set of ranges (default is 192.168.0.0/16,10.0.0.0/8,172.16.0.0/12)
- **any**: in this context, it means "from any source port", then there's an arrow '->' which means "a connection to"
- **msg**: is a directive that simply sets the message that will appear in the alert message sent/logged

alert: tells Suricata to report this behavior as an alert (it's mandatory in rules created for the STA).

tcp: means that this rule will only apply to traffic in TCP.

any: in this context, it means "from any source port", then there's an arrow '->' which means "a connection to" (there isn't a '<-' operator, but you can simply flip the arguments around the operator. You can use the '<>' operator to indicate that the connection direction is irrelevant for this rule), then an IP range which indicates the destination IP address and then the port. You can indicate a port range by using colon like 0:1024 which means 0-1024. In the round parenthesis, there are some directives for setting the alert message, metadata about the rule, as well as additional checks.

msg: is a directive that simply sets the message that will be sent (to Coralogix in the STA case) in case a matching traffic will be detected.

flow: is a directive that indicates whether the content we're about to define as our signature needs to appear in the communication to the server ("to_server") or to the client ("to_client"). This can be very useful if, for example, we'd like to detect the server response that indicates that it has been breached.

established: is a directive that will cause Suricata to limit its search for packets matching this signature to packets that are part of established connections only. This is useful to minimize the load on Suricata.

uricontent: is a directive that instructs Suricata to look for a certain text in the normalized HTTP URI content. In this example, we're looking for a url that is exactly the text "/root.exe".

nocase: is a directive that indicates that we'd like Suricata to conduct a case insensitive search.

classtype: is a directive that is a metadata attribute indicating which type of activity this rule detects.

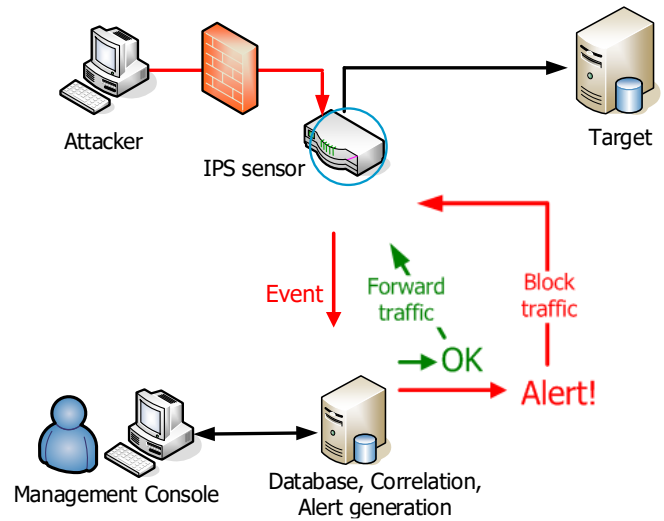
reference: is a directive that is a metadata attribute that links to another system for more information. In our example, the value url,<https://...> links to a URL on the Internet.

sid: is a directive that is a metadata attribute that indicates the signature ID. If you are creating your own signature (even if you're just replacing a built-in rule), use a value above 9,000,000 to prevent a collision with another pre-existing rule.

rev: is a directive that indicates the version of the rule.

There's a lot more to learn about Suricata rules which supports [RegEx](#) parsing, protocol-specific parsing (just like uricontent for HTTP), looking for binary (non-textual) data by using bytes hex values, and much much more. If you'd like to know more you can start [here](#).

- The traffic must flow through a component that can monitor and block traffic
 - When an event is triggered, the traffic that triggered it is blocked temporarily
 - A rule engine then determines whether an event is an alert
 - The rule engine might correlate it with other events from this or other sensors
 - If no alert is generated, the traffic is forwarded
 - If the event results in an alert, the traffic is blocked (logged and discarded)



- An IDS's performance can be assessed by measuring its false positive ratio and false negative ratio

$$FPR = \frac{FP}{N} = \frac{FP}{FP + TN}$$

"Ratio of benign data points classified as attack"

$$FNR = \frac{FN}{A} = \frac{FN}{TP + FN}$$

"Ratio of attack data points classified as benign"

N = #Data points that are not an attack
FP = #False Positives
IDS says «ATTACK» but it is none
TN = #True Negatives
IDS says «NO ATTACK» and it is none
A = #Data points that are attacks
TP = #True Positives
IDS says «ATTACK» and it is one
FN = #False Negatives
IDS says «NO ATTACK» but it is one

- A marketing sheet says:

- False positive ratio: 0.1%
- False negative ratio: 0%

That basically means the IDS does not miss any attack but that it reports some data points as “attacks” that aren’t attacks

=> **How good is this IDS?**

- Scenario 1:

- Data points analyzed per day: 500'000
- Data points that are attacks: 0.1%

- Analysis

- #Attacks = $500'000 \times 0.1\% = 500$
- #Benign = $500'000 - 500 = 499'500$
- From $FPR = \frac{FP}{N}$ it follows that
 $FP = FPR * N = 0,1\% * 499'500 \cong 499$
- Hence, it reports 500 real attacks and 499 “attacks” that aren’t attacks
- **~50% of alerts analyzed are attacks**

- A marketing sheet says:

- False positive ratio: 0.1%
- False negative ratio: 0%

That basically means the IDS does not miss any attack but that it reports some data points as "attacks" that aren't attacks

=> **How good is this IDS?**

- Scenario 2:

- Data points analyzed per day: 10'000'000
- Data points that are attacks: 0.006%

- Analysis

- #Attack = $10'000'000 \times 0.006\% = 600$
- #Benign = $10'000'000 - 600 = 9999400$
- From $FPR = \frac{FP}{N}$ it follows that
 $FP = FPR * N = 0,1\% * 9'999'400 \cong 9999$
- Hence, it reports 600 real attacks and 9999 "attacks" that aren't attacks
- **~5,7% of alerts analyzed are attacks**

- The false positive paradox is a statistical result where false positive tests are more probable than true positive tests
- This occurs when the overall population contains few attacks, and the attack rate is lower than the false positive rate

- Applications:
 - To monitor network or system activity to detect/prevent attacks and intrusion attempts
- Strengths:
 - Can help detecting known attack patterns (e.g., sql injection attempts, distributed scans, malware propagation, anomalous system behavior because of a system compromise,...)
 - IPS: If operated correctly, attacks are not only detected but stopped
- Limitations:
 - Must be configured and fine-tuned to have low FPR* and FNR in a specific environment
 - Network IDS cannot see through encrypted traffic (→ see notes)
 - An IDS does not stop an attack; it merely detects and logs it (→ human resources needed)
 - IPS: Packets must be buffered and can be delayed only briefly
 - Resource-heavy and limits correlation-possibilities with other events from the same or other sensors
 - IPS: False positives lock out legitimate users/activities

* Should be lower or similar to the attack rate in order to not to waste resources when analyzing alerts from the IDS.

NIDS and Encrypted traffic

There are two ways how to overcome this limitation:

1. Install network sensors on the host and intercept traffic before it gets encrypted.
2. Break up connections secured by TLS by performing a “man-in-the-middle attack”.

In the first case, one has to find a way to hook/modify the software components (usually system libraries) that implement the TLS layer on the system. However, since programs might come with their own implementation of TLS, not using system libraries, it is difficult to achieve full coverage. Furthermore, interfering with the network stack might have unwanted side-effects.

In the second case, all traffic is routed over a gateway that acts as a man in the middle. Hence, when a computer makes a connection to `www.example.com`, the connection is actually not with `www.example.com` but with the gateway. The gateway then makes a connection to `www.example.com` on behalf of the client and forwards data between those connections. To be able to act as `www.example.com` versus the client in case of a TLS connection, the gateway needs a certificate for `www.example.com` for which it knows the private key and which the client accepts as being valid. Since no public CA (=CAs which your operating system / browser trusts by default) would issue such a certificate to someone other than the owner of `www.example.com`, companies usually create their own CA and install the CA certificate on the companies computers. This way, the gateway can create certificates for any domain* which are considered valid by those computers. Since the TLS connection is now made with the gateway and then from the gateway to the target, the traffic is decrypted (and can be logged/inspected) and then sent to the target using

the TLS connection between the gateway and the target.

Note that performing TLS interception/inspection is a controversial topic as it basically undermines what TLS wanted to achieve. To learn more about the arguments of both sides, follow the debate on the Internet. You might start with a look at the current evolution of TLS with the most recent version TLS 1.3 making inspection/interception more difficult:

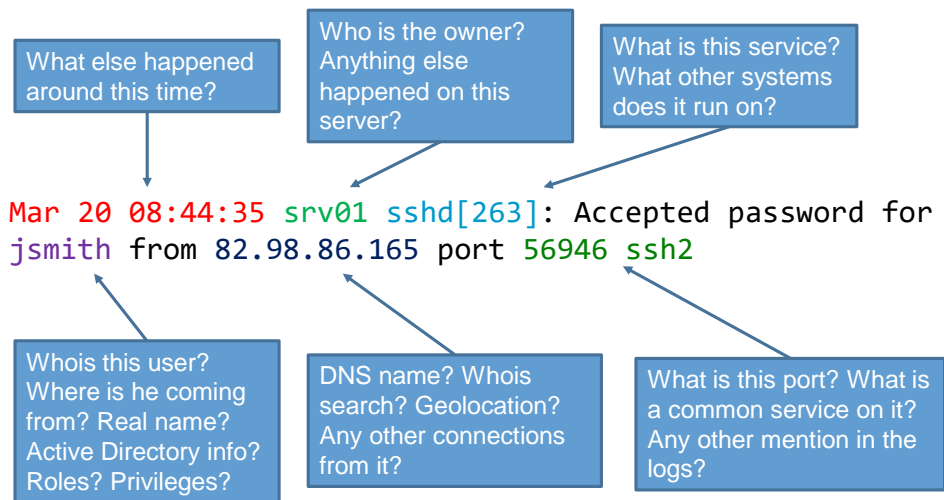
- <https://www.ncsc.gov.uk/blog-post/tls-13-better-individuals-harder-enterprises>

* There are some cases where this might not work. For example, if a program makes use of certificate pinning, i.e. has the public keys of (some) domains compiled in. The program then accepts a TLS connections to such a domain only, if the respective public key can be used to authenticate it. However, this means that the gateway would have to know the private key for this public key, which it can't.

Event Correlation

- Event Correlation – “Establish Relationships between events”
- Takes multiple isolated events, **combining** them into **a single relevant security incident** – “If this, followed by that, take some action”
 - Example: “Heartbleed vulnerability found on host x by Nessus followed by IDS alert of suspected Heartbleed attack” => raise alert
- **Comparative observations** based on **multiple parameters** such as:
 - Source/destination IP address
 - The user or machine involved
 - The time the activity began or ended
- Does **not rely on a single** IDS signature, device or rule
- Provide a very powerful type of **attack pattern analysis** based on **multiple independent inputs**

Key Task of a SIEM – Event Correlation (2)



- **Example #1:**

- Sally is a financial analyst who accesses PeopleSoft, email, billing system
- On Friday, logs show she has attempted to login to the DNS server of the company over ssh
- Sally logs into the Active Directory between 8:30am and 8:39am Mo.-Fr.
- This Saturday/Sunday, Sally logs into the AD at 1:31am, 6:02am, 8:33pm

- **Example #2:**

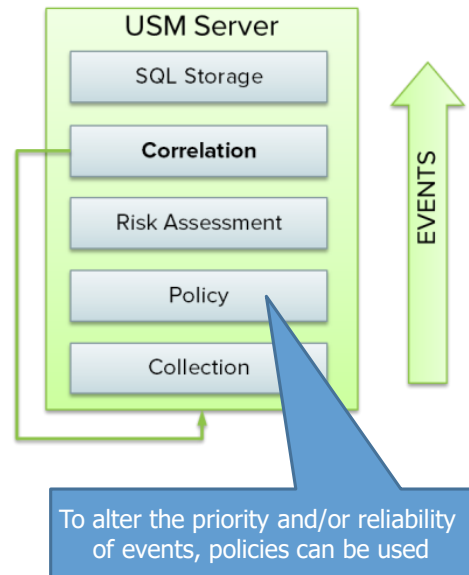
- Your SIEM logs physical access (through your ID badge system)
- On Tuesday at 4:00pm Joe Smith swipes his card exiting the HQ building
- On Tuesday at 4:02pm Joe Smith login in to a server on the perimeter of your network

- **Question:**

- Baseline: What and how to establish?
- Anomaly: What and why?
- Correlation: What and why?

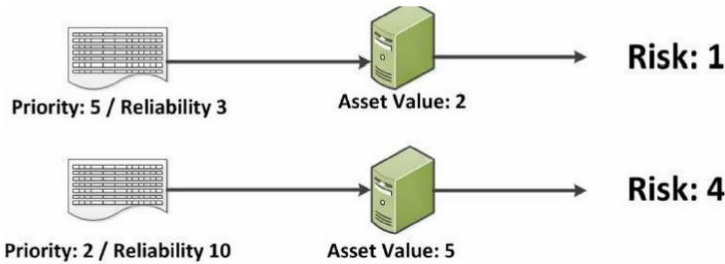
=> Correlation and "clean" baselines
are very important!

- First, **normalized** events are received
- Next, they are evaluated against **policies** consisting of **conditions** and **consequences**
 - Conditions: Determine **which events** are processed by a policy
 - Consequences: Define **what will happen** when events match a specified condition
 - Send Email
 - Forward/modify/create new/drop event
 - ...



```
risk=asset * (reliability * priority / 25)
```

Higher Risk Scores help the analyst know what to examine first!



What do you think about this formula?

Alarm: Risk >=1

- **Asset value:**
 - Basis: Asset value ☺
 - From 0 to 5
- **Priority value:**
 - Basis: Event type
 - From 0 to 5
- **Reliability value**
 - Basis: Is an attack?
 - From 0 to 10.

- **Event Priority:** Default value 2. This is a priority ranking (0 to 5) that is based on the event type, such as authentication failure, web attack, or denial of service, which indicates the urgency with which an event should be investigated. (AlienVault provides an event taxonomy to classify various events by category and subcategory. See [USM Appliance Event Taxonomy](#)).
- **Event Reliability:** Default value 2. This is a reliability ranking (0 to 10) that specifies the likelihood that an event is a real attack or a false positive event.
- **Asset value:** Default value 2. The “value” of the targeted asset. This is difficult to set in general. It might even change over time, depending on what a certain target is currently running/storing.

Defining an assets' value: A very simple and imperfect way to do this could be as follows.

- 5: For any server/device which can receive a packet from the Internet, or has unencrypted access to valuable data (PCI, bank, PHI, SSN, etc). Any Domain Control, LDAP server, or any other form of authentication services device. VPN device.
- 4: Any database server which did not falls into the above. App servers Source code repo's.
- 3: Any other Prod server or devices
- 2: Any non-prod devices
- 1: any server or device which really don't care about

Source: https://www.alienvault.com/doc-repo/usm/security-intelligence/AlienVault_Life_cycle_of_a_log.pdf

- What about **vulnerability scanning results**?
- OpenVAS results can be turned into a source of events*
 - The OpenVAS plugins appear as distinct sources of «vulnerability» events
 - Priority can be tuned for each source
 - Correlation is done on events
 - Severity of vulnerability is mapped to reliability

* Vulnerability scanning results are not fed into the «events» pipeline in the AlienVault SIEM by default, it can be modified accordingly.
How this can be done is described here: <http://timbrigham-sbn.tumblr.com/page/2>

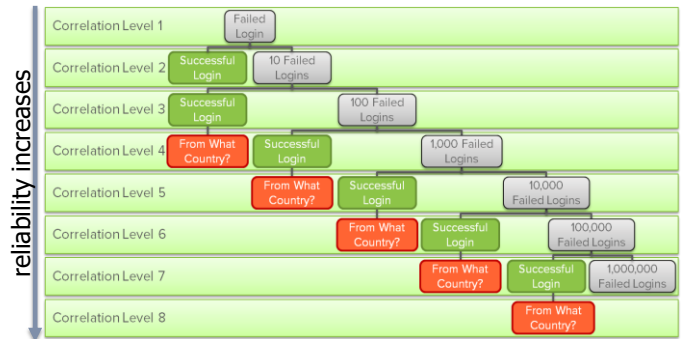
- Correlation and cross-correlation

- Correlation:

- A correlation rule typically associates multiple events, of the same or different event types, from the same data source
- Correlation directives, which contain one or more correlation rules, decide whether or not to connect certain events

- Example: Detecting brute force authentication events

- Correlates "failed login" and "successful login" events from the same IP
- Can be limited to hosts with bad reputation (country of origin, blacklists,...)



Detecting brute force authentication:

This directive detects brute force authentication events by connecting two types of events, Failed Logins and Successful Login. Based on the number of occurrences of individual events, the correlation engine can conclude that the event represents a case of an administrator mistyping a password (one failed login attempt followed by a successful login), a successful brute force attack with low reliability (10 failed login attempts followed by a successful login), or a successful brute force attack with high reliability (100,000 failed login attempts followed by a successful login). All these events need to come from the same IP address and go to the same IP address, in order for the directive event to be created. The correlation engine can also take into account the reputation of source and destination IP addresses, such as Country of Origin, and match specific rules only if an event is coming from, or destined to, a host with bad a reputation.

Source: <https://www.alienvault.com/documentation/usm-v5/correlation/about-correlation.htm>

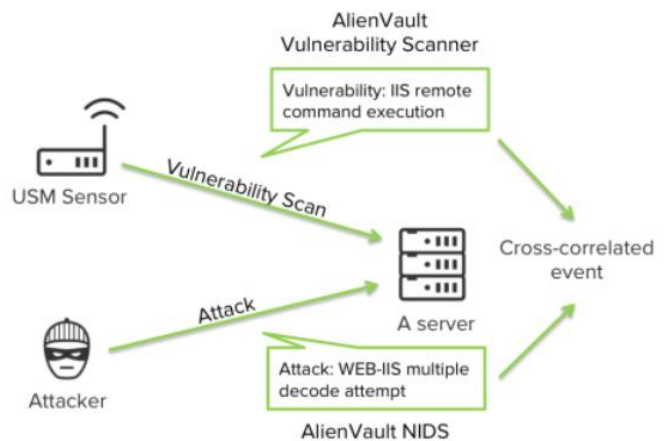
AV Network attack, too many dropped inbound packets from DST_IP
Environmental Awareness, Network Anomaly, Too many dropped inbound packets - Priority 2

RULES

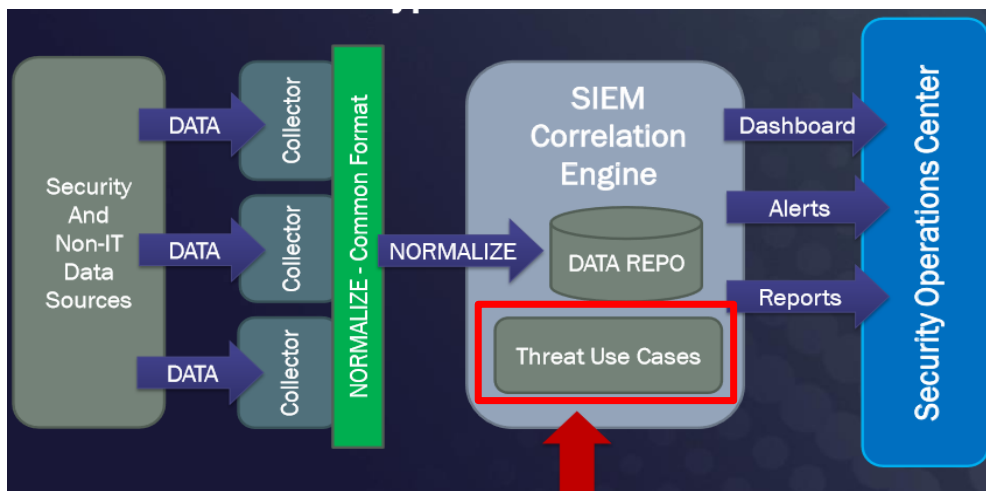
	NAME	RELIABILITY	TIMEOUT	OCCURRENCE	FROM	TO
	▼ Firewall dropped packet	2	None	1	ANY	ANY
AND →	▼ Firewall dropped packets	4	10	3	ANY	1:DST_IP
AND →	▼ Firewall dropped packets	6	20	5	ANY	1:DST_IP
OR →	Firewall dropped packets	8	30	10	ANY	1:DST_IP

Can this event trigger an alert?

- **Cross-Correlation** – Correlation of events from different data sources
- **Example:**
 - Correlate IDS alerts with relevant vulnerability information
 - Modifies reliability tag of an event to the maximum, if target is vulnerable



(Threat) Use Case

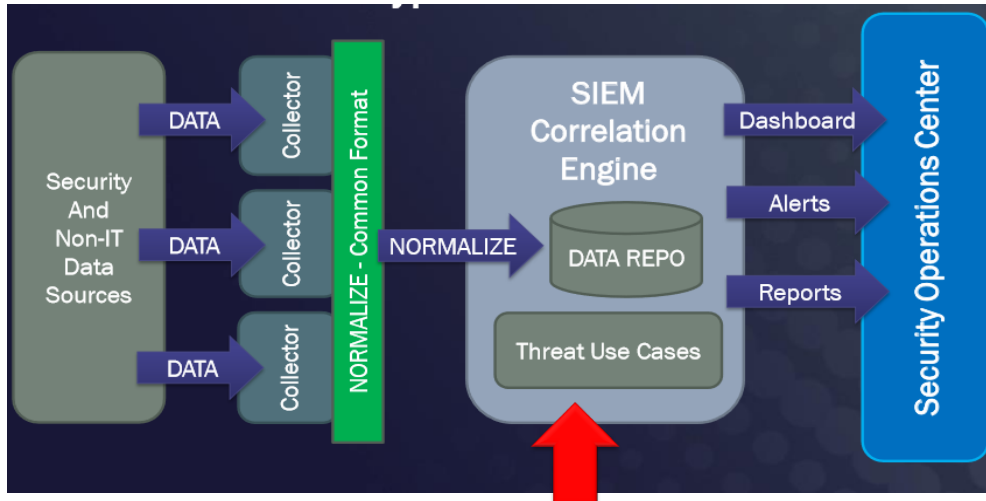


- The **key for a SIEM to be useful** and manageable is the definition and implementation of **use cases** - a **logical**, **actionable** and **reportable** component
- It can be either a **(set of) rule(s)**, **report**, **alert** or **dashboard** which solves a set of needs or requirements
 - E.g., rules to detect outbound spam
 - E.g., a weekly report on the known vulnerabilities
 - E.g., a dashboard element showing successful logins from "bad country"
- Use cases are **the heart of any SIEM deployment** and very **time-consuming** - SIEM don't work (well) out of the box!
- A use case is **"developed"** – like a mini project it has several stages
 - See the appendix for a very high-level example to get an idea

See the appendix for a bit more info on the various stages for the example "outbound spam detection".

In brief, the stages are:

1. Requirements definition
2. Definition of the scope
3. List of data & event sources
4. Validation that data is available
5. Define the logic (rules, report generation, dashboard views,...)
6. Implementation and testing
7. Definition of use case response
8. Maintenance

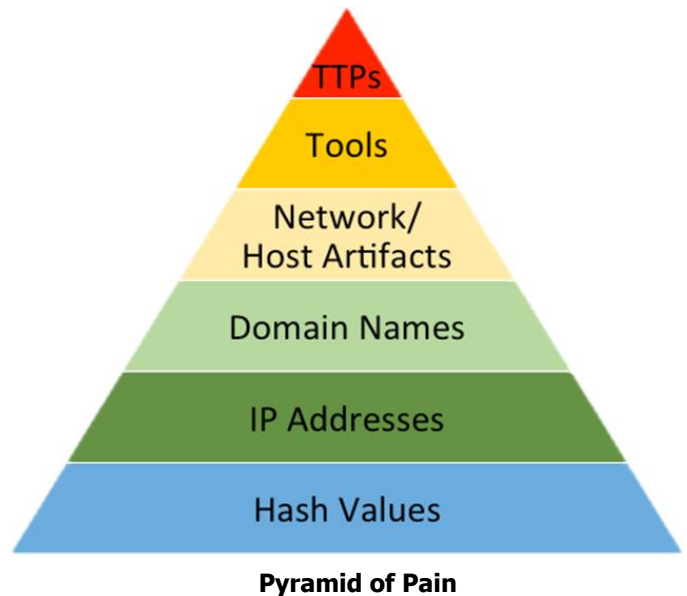


Machine-Readable Threat Intelligence
Threat Feeds with Indicators of Compromise

Indicators (of Compromise)

The Pyramid of Pain

- Being able to **detect and respond to an indicator** quickly enough means **denying the adversary its use**
 - *Hashes, IP addresses, domain names, network/host artifacts, tools, tactics techniques and procedures (TTPs)*
- Some indicators are **much better than others** - they cause **more pain** to the adversaries if they must tune an attack to evade such an indicator
- The **Pyramid of Pain** illustrates this concept of indicators of different value



Source: <http://detect-respond.blogspot.gr/2013/03/the-pyramid-of-pain.html>

- SHA1, MD5 or other similar **hashes** that correspond to specific suspicious or malicious files
- Accuracy – **High**
 - Same hash for different files is almost impossible
- Pain level – **Very low**
 - Flipping a single bit is enough

Note that the pain level can be high or very high in the following cases:

- In case of **whitelisting** to mark anything that is not on the list as suspicious, the pain level is **very high** => needs finding a hash collision!
- In case **fuzzy hashes** are used, the level of pain is **significantly higher** since adapting to them is challenging

Unfortunately, a purely whitelisting based approach is rarely an option on general purpose systems since data files are anyway hard if not impossible to whitelist. They are subject to change but might still contain executable code (Macros, OLE objects,...).

Fuzzy hashes: Two files with only minor or moderate differences would have fuzzy hash values that are substantially similar, allowing an investigator to note a possible relationship between them.

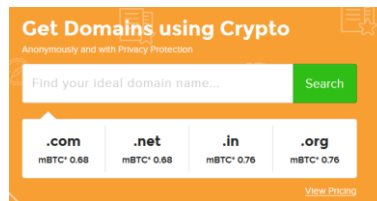
- **Ssdeep** (<http://ssdeep.sourceforge.net/>) is an example of a tool commonly used to compute fuzzy hashes.
- Fuzzy hashes fit better the "Tools" level of the Pyramid because they are more resistant to change and manipulation.
- In fact, the most common use for them is to identify variants of known tools or malware.

- An IP address or a subnet or a set of IP addresses
- Accuracy – Medium-High
 - Same IP used by legitimate services/people unlikely but possible
- Pain level – Low
 - Attacker <=> Victim – (Anonymous) proxy services can be used
 - Victim => Attacker – Fast-fluxing technique can be used

Fast-Flux:

- For victim to attacker communication. A domain name is used to lookup the IP. The IP to which the domain maps is changed frequently.

- A **domain name** (e.g., "evil.net") or a sub- or sub-sub-domain (e.g., "this.is.sooooo.evil.net"), maybe even a hijacked-one*
- Accuracy – **Medium-High**
 - Domain name also **used by legitimate services/people** unlikely but **possible**
- Pain level – **Low-Medium**
 - Must be **registered and paid** for (even if with stolen funds)
 - Must be hosted somewhere (at least: DNS entry)
 - The large number of providers with **lax registration standards** don't make this a big issue
 - **Anonymous** domain registration
 - Example: <https://bitdomain.biz/>



*: Subdomain Hijacking

Or you might want to use the good reputation of an already existing domain? One scheme to do this, is the scheme called **Subdomain Hijacking**.

It works as follows: Hijack/takeover attacks can happen when a company creates a DNS entry that points to a third party service (CNAME Record) such as:

sampleapp.example.com d111111abcdef8.cloudfront.net/sampleapp

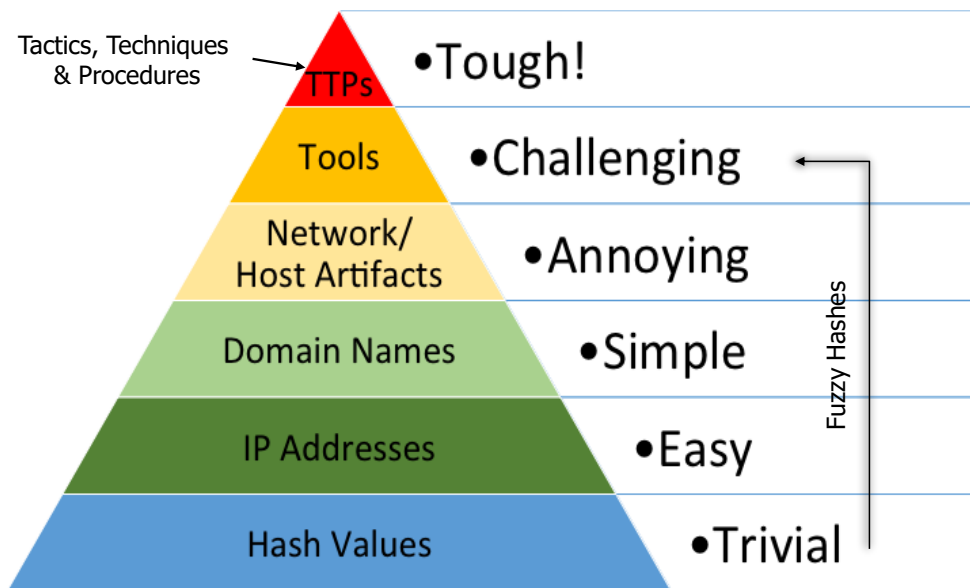
and then discontinues this site or application but does not remove the DNS entry. In this case, an attacker could register/claim the domain to which the entry points to and setup and run its malicious service there. The service would seem to be a service from example.com but it isn't.

- **Observables** caused by adversary activities **on your host/network** that tend to distinguish malicious activity from that of legitimate users
 - **Network:** URI patterns, C2 information embedded in network protocols, distinctive HTTP User-Agent or SMTP Mailer values, etc.
 - **Host:** Registry keys/values known to be created by a piece of malware, files or directories dropped in certain places or using certain names, names or descriptions or malicious services
- **Accuracy – Medium-High**
 - Depends on the artifact used
- **Pain level – Medium**
 - Attackers must find out **why they are detected**
 - Attackers must find out **how to fix** and then **fix it**
 - Example: HTTP recon tool
 - A mistake in the User-Agent string (excess space or semicolon) => indicator

- **Multiple host and/or network artifacts** of a tool used by the adversary to accomplish their mission can be identified
 - Utilities designed to create malicious documents for spear phishing, backdoors used to establish C2, or other host-based utilities used after compromising the target
- **Examples**
 - **Network** - Rules capturing (custom) communication protocols (structure, message types,...)
 - **Hosts** - Rules capturing multiple host artifacts of the tool
 - **Fuzzy hashes** of files (or memory content) used by the attacker
- **Accuracy – High** (multiple artifacts is more accurate than a single artifact)
- **Pain level – High**
 - A **new tool** for the same purpose **must be created** -
 - The research, development and training required costs time/money!

- TTP: **How the adversary goes about accomplishing their mission**
- Examples of TTPs (attack vectors):
 - Dumping cached authentication credentials and reusing them (e.g., Pass-the-Hash attack)
 - Spear phishing for establishing a presence in the network
 - **Tailgating** to get access to a building
- Accuracy – **Low-High**
 - Depending on the characteristics used to identify a TTP
- Pain level – **Very High**
 - Operate on **adversary behaviors (attack vectors)**, not against tools
 - Detect Pass-the-Hash attacks themselves rather than the tools used for it
 - Forces adversaries to **learn new behaviors** (attack vectors)
 - Attackers have only two options: Give up or **reinvent themselves** from scratch.

Another example is **social engineering**. If people know about the TTP “**tailgating**” to enter a building without the required access key(s), people always check (in the ideal world...) that no one slips through the door without having proven that he/she possesses a valid access key. The adversary now has to **figure out a new TTP to get access to the building**.



Source: <http://detect-respond.blogspot.gr/2013/03/the-pyramid-of-pain.html>

The Pyramid of Pain

The Pyramid of Pain illustrates this concept of indicators of different value. It shows the relationship between the types of indicators you might use to detect an adversary's activities and how much pain it will cause them when you are able to deny those indicators to them.

Types of Indicators

- **Hash Values:** SHA1, MD5 or other similar hashes that correspond to specific suspicious or malicious files. Often used to provide unique references to specific samples of malware or to files involved in an intrusion.
- **IP Addresses:** An IP address or a subnet or a set of IP addresses
- **Domain Names:** A domain name (e.g., "evil.net") or a sub- or sub-sub-domain (e.g., "this.is.sooooo.evil.net")
- **Network Artifacts:** Observables caused by adversary activities on your network that tend to distinguish malicious activity from that of legitimate users. Examples are URI patterns, C2 information embedded in network protocols, distinctive HTTP User-Agent or SMTP Mailer values, etc.
- **Host Artifacts:** Observables caused by adversary activities on one or more of your hosts that tend to distinguish malicious activities from legitimate ones. Examples are registry keys or values known to be created by specific pieces of malware, files or directories dropped in certain places or using certain names, names or descriptions or malicious services or almost anything else that's distinctive.
- **Tools:** Software used by the adversary to accomplish their mission. Examples are utilities designed to create malicious documents for spearphishing, backdoors used to establish C2 or password crackers or other host-based utilities they may want to use post-compromise.
- **Tactics, Techniques and Procedures (TTPs):** How the adversary goes about accomplishing their mission, from reconnaissance all the way through data exfiltration and at every step in between. "Spearphishing" is a common TTP for establishing a presence in the network. "Spearphishing with a trojaned PDF file" or "... with a link to a malicious .SCR file

disguised as a ZIP" would be more specific versions. "Dumping cached authentication credentials and reusing them in Pass-the-Hash attacks" would be a TTP.

The Pyramid Explained

Now that we have a better idea what each of the indicator types are, let's take a look at the pyramid again. The widest part of the pyramid is colored green, and the pinnacle of the pyramid is red. Both the width and the color are very important in understanding the value of these types of indicators.

Hash Values

On the one hand, hash indicators are the most accurate type of indicator you could hope for. The odds of two different files having the same hash values are so low, you can almost discount this possibility altogether. On the other hand, *any* change to a file, even an inconsequential one like flipping a bit in an unused resource or adding a null to the end, results in a completely different and unrelated hash value. It is so easy for hash values to change, and there are so many of them around, that in many cases it may not even be worth tracking them.

Fuzzy hashes: Two files with only minor or moderate differences would have fuzzy hash values that are substantially similar, allowing an investigator to note a possible relationship between them.

[Ssdeep](http://ssdeep.sourceforge.net/) (<http://ssdeep.sourceforge.net/>) is an example of a tool commonly used to compute fuzzy hashes. Fuzzy hashes fit better the "Tools" level of the Pyramid because they are more resistant to change and manipulation. In fact, the most common use for them is to identify variants of known tools or malware.

IP Addresses

IP addresses are quite literally the most fundamental indicator since almost all attacks/compromises need at some point a network connection to carry out an attack or data exfiltration action. It's at the widest part of the pyramid because there are just so many of them. If you deny the adversary the use of one of their IPs, they can usually recover by using another one without even breaking stride. Tools like anonymous proxy service (e.g. Tor) help them to change IPs quite frequently and transparently. That's why IP addresses are green in the pyramid.

Domain Names

One step higher on the pyramid, we have Domain Names (still green, but lighter). These are slightly more of a pain to change, because

- they must be registered and paid for (even if with stolen funds)
- they must be hosted and hosted somewhere
- it takes some time for the new domains to become visible throughout the Internet (up to a day or two)

However, the large number of DNS providers out there with lax registration standards (many of them free) don't make this a big issue.

Network & Host Artifacts

Detecting and responding to indicators at this level, causes the attacker to go back to their lab and reconfigure and/or recompile their tools. If an attacker's HTTP recon tool uses e.g., a distinctive User-Agent string when searching your web content (off by one space or semicolon, for example) and you block any requests which present this User-Agent, you force them to go back and spend some time

- a) figuring out how you detected their recon tool, and

- b) how to fix it and then fix it

Tools

If we get so good at detecting the artifacts of their tool in so many different ways, we might force them to find or create a new tool for the same purpose, if they are able to. The research, development and training required just cost them some real time and money.

Some examples of tool indicators might include Yara (<https://plusvic.github.io/yara/>) signatures as used by many anti-malware service provider. With YARA you can create descriptions of malware families (or whatever you want to describe) based on textual or binary patterns. Each description, a.k.a rule, consists of a set of strings and a boolean expression which determine its logic.

Network aware tools with a distinctive communication protocol may also fit in this level, where changing the protocol would require substantial rewrites to the original tool. Also, as discussed above, fuzzy hashes would probably fall into this level.

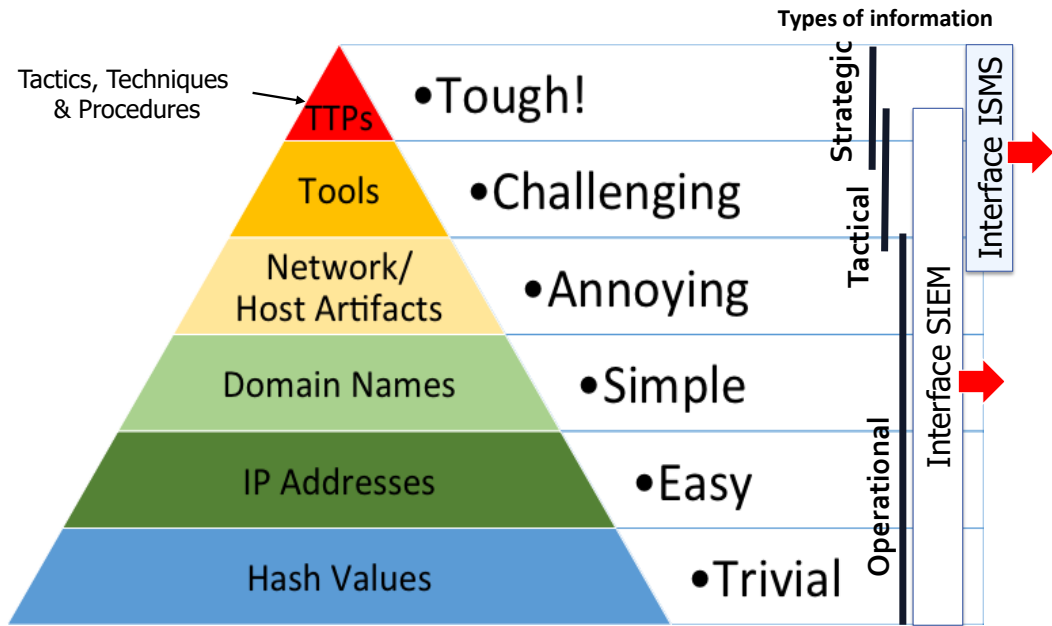
Tactics, Techniques & Procedures

At this level, you are operating directly on adversary behaviors, **not** against their tools. For example, you are detecting Pass-the-Hash attacks themselves (perhaps by inspecting Windows logs) rather than the tools they use to carry out those attacks. From a pure effectiveness standpoint, this level is your ideal. If you are able to respond to adversary TTPs quickly enough, you force them to do the most time-consuming thing possible: **learn new behaviors** (attack vectors).






















When you are able to do this across a wide variety of the adversary's different TTPs, you give them one of two options: Give up, or **Reinvent themselves from scratch**.

Source (slightly modified): <http://detect-respond.blogspot.gr/2013/03/the-pyramid-of-pain.html>

Security Monitoring – What should we look for?



Source: <http://detect-respond.blogspot.gr/2013/03/the-pyramid-of-pain.html>

Detection Activity	Threat Feed Data			Real Time Data
Detect Outbound Browser Activity to known CC & Malware drop sites IP/Domains	 Domain	 URI	 IP Addr	Web Proxy Logs, Firewall Logs
Detect Outbound Network Activity to known CC & Malware drop sites IP/Domains	 Domain	 URI	 IP Addr	Firewall Logs, Netflow Traffic
Detect Inbound Network Activity from CC & Malware drop sites IP/Domains	 Domain	 IP Addr		Firewall Logs, Webserver Logs
Detect Outbound Network Activity to Dyn-Dns Hosts	 DYN-DNS			DNS Logs, Firewall Logs
DNS Queries to non standard TLD	 DNS	 TLD		DNS Logs
Inbound Phishing attempts against organization web servers	 Domain	 IP Addr		Web Server Logs
Email from Known Phishing Servers	 E-mail			SMTP Logs, Exchange Logs
Infected User Accessing a Critical Asset	 Domain	 URI	 IP Addr	Suspicious User Active List
Accepted Outbound Firewall Connections to Identified Malicious sites	 Domain	 IP Addr		Firewall Logs
Infected Host has not triggered an Anti Virus Alert	 Domain	 IP Addr		Anti-Virus

- **Log management systems** should have a **scalable** and **reliable** architecture to handle large amounts of log data - **millions to >>billions** of events per day
- **SIEM functionality** is needed to being able to find a **needle in a stack of needles**. Stealthy attacks require making use of all possible traces there are...
- **Correlation rules** can be as simple as correlating several log messages of the same type (aggregation) to increase the reliability of an alert or complex ones involving **multiple sources** and **conditions**
- An **asset inventory** can provide information to **prioritize** alerts (e.g., value of assets) and a **vulnerability scanner** can provide information that helps to **reduce false positives** when **correlated** with alerts
- The **Pyramid of Pain** shows the way how to make life difficult for attackers – **use indicators that cause the most pain** whenever possible

Appendix

- Use case: A **logical**, **actionable** and **reportable** component of a SIEM
- It can be a **rule**, **report**, **alert** or **dashboard** solving a set of needs/ requirements
- A use case is “**developed**” – like a mini project it has several stages

1. Requirements definition

- Business, compliance, regulatory, security

2. Definition of the scope

- Typically the IT infrastructure that is a high priority for the specific requirement

Example:

Outbound Spam Detection

Mail infrastructure, machines of end users, security monitoring infrastructure

3. List of the event sources

- Definition of the data required for this use case (log, alert data,...) from the systems in the defined scope

4. Validation phase

- Making sure that the data defined in step 3. is available or can be made available and is parsed properly

• Relevant sources:

- HIDS/NIDS (signature-based)
- Mail hygiene/filtering tools (signature)
- Events from network devices (traffic anomaly-based detection)
- Events from endpoint detection tools (signature + traffic anomaly-based)

• The fields to validate/check

- Source IP, source user ID, Email addresses, target IP, host information of source and target, event names for SPAM detection, Port and Protocol for SMTP based traffic detection etc.

5. Define the logic

- Exact definition of what and how much data is needed along with the attack vector that should be detected
- Signature or behavior-based logic
 - One machine doing Port 25 Outbound connections at the rate of 10 in a minute
 - SPAM Signatures originating from the same source from IDS/IPS, Mail Filter etc. having the same destination public domain
 - SYN Scans on port 25 from a single source
 - ...

6. Implementation and testing

- Define the desired output and configure the SIEM to do the correlation + alerting
- Output e.g., a report, a real-time notification, a retrospective notification

7. Define use case response

- Procedures that help to make the use case operational

8. Maintenance

- Define ongoing process to keep the use case relevant by appropriate tuning

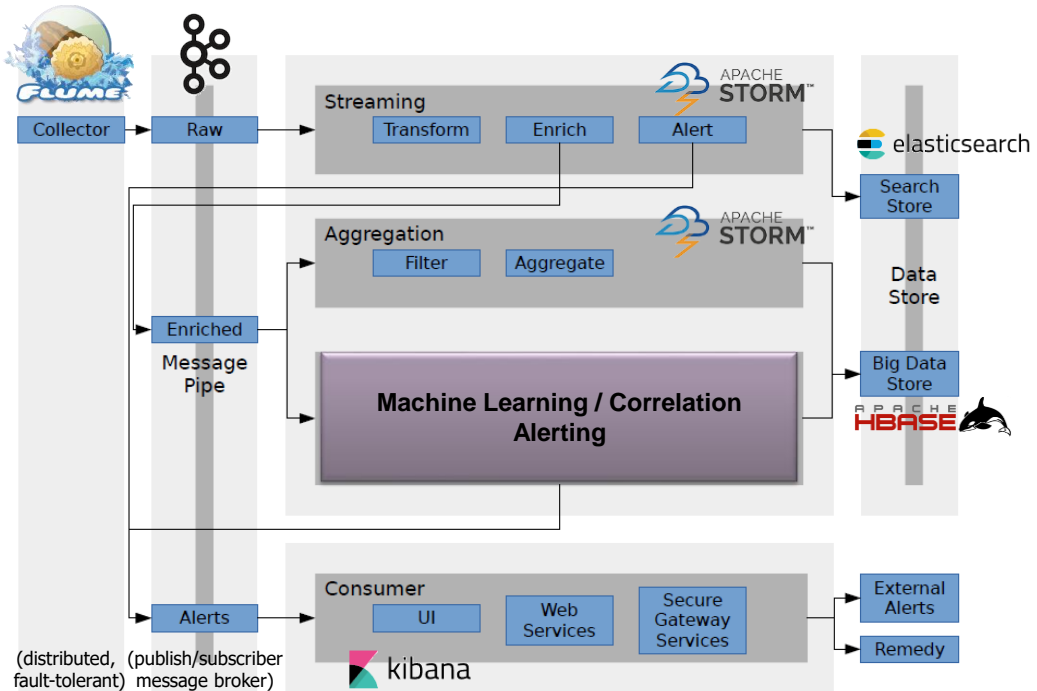
• Tuning: Improve accuracy

- Several iterations of incident analysis and data collection => ensure that use case is doing what it should do!
- Threshold adjustment, logic tightening,...

• Investigate SPAM cases – use guides/checklists for this.

- Adapt and tune. Delete if there is no SPAM anymore in the world... 😊

Log Management – Sample Architecture (including alerting)



This is one of many possible (high-level) architectures for collecting, aggregating/enriching, storing and processing (e.g., for anomaly detection) log data from any source.

The technologies used are technologies we used for building a monitoring system in a CTI project for a startup in the cloud provider business.

1. Data Collector (Apache Flume):

Apache Flume is a distributed, reliable collector for large amounts of data with focus on availability and is therefore suitable as entry point to the log management core. As Apache Flume has a master/slave architecture, it manages itself and can reorganise the cluster in case of failed nodes. Apache Flume is very fault tolerant and reduces possible data loss to a minimum.

Flume sources listen for and consume “log events”. “Log events” can range from newline-terminated strings in stdout to HTTP POSTs and RPC calls — it all depends on what sources the agent is configured to use. Flume agents may have more than one source, but must have at least one. Sources require a name and a type; the type then dictates additional configuration parameters. On consuming an event, Flume sources write the event to a channel (here: the Kafka message broker). Importantly, sources write to their channels as transactions. By dealing in events and transactions, Flume agents maintain end-to-end flow reliability.

2. Data “Routing” (Apache Kafka):

Apache Kafka is a publish/subscribe message broker. It can be elastically and transparently expanded (with new nodes/resources) without downtime. Apache Kafka brings the possibility to divide the pipe into topics to hold messages from different stages from the streaming on one topic.

3. Data Normalisation/Enrichment and real-time alerting (Apache Storm)

The streaming part is where the transformation, normalisation and the enrichment of the gathered data takes place. The transformation and normalisation part is responsible to bring all the data in a common format, so that correlation and rating, scoring and filtering will be easier. Most likely the

data will be transformed into JSON format, as this is easy machine and human readable and gives us the possibility to add nested information. The enrichment part is a possibility to add valuable information to the gathered data. Some examples are e.g., adding geolocation and DNS information for IP addresses.

It is also the part where the real-time monitoring happens and alerts will be sent immediately (e.g., communication with blacklisted hosts).

This part of the core system will store all data passing through into a search store (**Elasticsearch**).

For information about Apache Storm, see step 4 below.

4. Aggregation (Apache Storm)

In the aggregation part the data is filtered, correlated and aggregated to reduce the amount of data (long-term storage in Big Data Store).

For example, only data from certain systems is retained or the number of failed logins (in a certain timespan) of a user instead of the full loglines.

Apache Storm is a distributed real-time computation system widely used for this purpose (e.g., Yahoo, twitter and Spotify). It consumes streams of data and processes those streams in arbitrarily complex ways. Apache Storm allows to implement stream processing on the cluster with so called bolts. Bolts will be distributed over the cluster and the implemented algorithms are fault tolerant and provide scalability.

For the **big data store**, Apache HBase could be used. It provides Bigtable-like capabilities on top of Hadoop and HDFS. It provides a fault-tolerant way of storing large quantities of sparse data. Hbase provides a fast way to find small amounts of data within a large collection of information.

5. Machine learning / Correlation / Alerting

Here, the actual “intelligence” can be found. This is where (at the moment) no real open source solutions exist.

Note also, that this step could also be done solely on data in the search store or big data store or by taking this data into consideration. However, since queries might take some time, alerts are not real-time.

6. Visualization Dashboards (Kibana)

Kibana provides a generic UI for searching and visualising data. It can be used to build dashboards to analyse and compare time series (and other types of data). Kibana is a client side only application built in Java Script, which queries data directly from Elasticsearch or other data sources.

An interesting article with some insights in options and pros and cons of some of the technologies in our sample architecture:

<http://accelazh.github.io/elasticsearch/Logstash-Elasticsearch-Kibana-Investigation>