

Malware Detection: Static vs Dynamic



WARNING



- ❑ HUGE and EXTREMELY COMPLEX topic
- ❑ We will only scratch the surface of the surface
- ❑ Focus of this section: **EDR** on a **single** node (in **isolation**)

Malware detection (REMINDE)



- ❑ Identification of **malicious software**
- ❑ Typically done on **endpoints**
- ❑ Detection **before** or **during** execution
- ❑ Focus on **prevention**

Static vs Dynamic



- ❑ Identification of **malicious software**
- ❑ Typically done on **endpoints**
- ❑ Detection **before** or **during** execution
- ❑ Focus on **prevention**

- ❑ **Static** analysis (AV)
 - ❑ Scan of **files/memory content** in search of **predefined signatures**
- ❑ **Dynamic** analysis (AV/EDR)
 - ❑ Analysis of **execution events** in search of **predefined behaviors**

Scanning



- ❑ **Signature-based scanning**

- ❑ Hundreds of thousands of signatures
($\approx 200K$ in Windows Defender)

- ❑ **Files**

- ❑ **On-access:** Download, Open, Delete
("Real-time protection" in Windows Defender)


- ❑ **Memory**

- ❑ When a process is created and a **file** is mapped in memory

Scanning Bypass



- ❑ **Signature**-based scanning
 - ❑ Restructure the file content before using it so that:
 - ❑ **Functionality** preserved
 - ❑ Signature **not** matched
 - ❑ Polymorphism ("each sample is different")

 - ❑ Not so simple in practice, but done more or less routinely
- 
- ❑ We need a **further** line of defense: **execution (behavior)**

Dynamic (Behavioral) Analysis

- ❑ **Dynamic** analysis (AV/EDR)
 - ❑ Analysis of **execution** events in search of **predefined behaviors**
- ❑ **What** can be analyzed?
 - ❑ Memory accesses
 - ❑ Machine instructions?
- ❑ What it means "**behavior**", exactly?



What can be analyzed: EDR Sensors



1. System call invocations

2. Predefined events created by O.S. or applications

- ☐ Activity related to processes, memory, files, network

- ☐ Each sensor output ("event") has a lot of information:
 - ☐ Process ID, parent process ID, process image, ...
 - ☐ File / Message content, ...
 - ☐ ...

What can be analyzed: EDR Sensors



- ❑ EDR receives sensor outputs ("events") as:
 - ❑ Logging
 - ❑ Callback-before // May **block** suspicious operation
 - ❑ Callback-after

- ❑ Analysis of events may detect suspicious behavior

Behavioral analysis in practice



Example: Self-Injection

- ❑ Write malicious program M
- ❑ Compile M and obtain sequence of instructions $I(M)$ (**shellcode**)
- ❑ Write program M1 that, **during execution**:
 1. Allocate a memory page P with RWX access rights
 2. Declare a variable initialized with $I(M)$
 3. Copy $I(M)$ to P
 4. Jump to $I(M)$
- ❑ Execute M1 on target

M1 **injects** code
into itself


Hmmm...

- ❑ Write program M1 that, **during execution**:
 1. Allocate a memory page **P** with RWX access rights
 2. Declare a variable initialized with I(M)
 3. **Copy I(M) to P**
 4. Jump to I(M)
- ❑ Execute M1 on target

- ❑ I(M) could match a signature
- ❑ **Memory scanning on P** could detect this malicious behavior



Memory Scanning



❑ Memory

- ❑ When a process is created and a **file** is mapped in memory
- ❑ While a process is **running**, its memory is **not** scanned
 - ❑ It might be scanned, but it is not: excessively costly
- ❑ It suffices that process memory looks **innocuous upon process creation**

Self-Injection: Suspicious Behavior

```
const char str1[] = .../* shellcode */  
const size_t lenstr1 = sizeof(str1);  
PVOID runIt = VirtualAlloc(0, lenstr1, ..., PAGE_RWX);  
memcpy(runIt, str1, lenstr1);  
CreateThread(..., runIt,);
```

- ❑ No malicious code at process creation time
- ❑ Self-injection is a **suspicious behavior**:
What you **inspected** at load time <> what you **execute**
- ❑ What you do might not be malicious...but this behavior is

Hmmm...

- ❑ Write program M1 that, **during execution**:
 1. Allocate a memory page P with RWX access rights
 2. Declare a **variable initialized with I(M)**
 3. Copy I(M) to P
 4. Jump to I(M)
- ❑ Execute M1 on target

- ❑ I(M) could match a signature
- ❑ **Memory scanning on data section of M1** could detect this malicious behavior



Encrypted shellcode

```
const char str1[] = .../* encrypted shellcode */
#define XOR_KEY = 0xAA
for (DWORD i = 0; i < size; i++) {
    str1[i] ^= str1[i] ^ XOR_KEY; // bitwise XOR
}
const size_t lenstr1 = sizeof(str1);
PVOID runIt = VirtualAlloc(0, lenstr1, ..., PAGE_RWX);
memcpy(runIt, str1, lenstr1);
CreateThread(..., runIt,);
```

- ❑ Most malware contains **encrypted** shellcode that is decrypted at runtime
- ❑ One of the many ways for defeating scanners

Self-Injection: How to detect?

```
const char str1[] = ...
/* decryption */
...
const size_t lenstr1 = sizeof(str1);
PVOID runIt = VirtualAlloc(0, lenstr1, ..., PAGE_RWX);
memcpy(runIt, str1, lenstr1);
CreateThread(..., runIt,);
```

- ❑ How to **detect** this specific **behavior**?
- ❑ How to detect it with high precision
(**no false positives**)?



Detecting Self-Injection (I)

```
const char str1[] = .../* shellcode */  
const size_t lenstr1 = sizeof(str1);  
PVOID runIt = VirtualAlloc(0, lenstr1, ..., PAGE_RWX);  
memcpy(runIt, str1, lenstr1);  
CreateThread(..., runIt,);
```

- ❑ Creation of memory pages with RW access rights is very commonplace (dynamically allocated memory)
- ❑ But with **execution** access rights? Why?
- ❑ **Detection rule:** process allocates memory page with WX access rights

Detecting Self-Injection (II-a)

- ❑ **Legitimate** usages of self-injection: **Interpreted** languages
 - ❑ Javascript, Python, Java, ...
- ❑ We do not want to trigger the detection for them
- ❑ More precise Detection rule:
`process allocates memory page with WX access rights`
AND
`process.name is not ("Chrome.exe", "Edge.exe", "python.exe", ...)`

Detecting Self-Injection (II-b)

- ❑ `process` allocates memory page with WX access rights
AND
`process.name` is not (...)
- ❑ Let us assume we have an **exhaustive** list with **full** pathnames
 - ❑ Relative pathnames can be circumvented easily

Practical Scenario



❑ process allocates memory page with WX access rights
AND
process.name is not (...)

1. A **new** program that uses Javascript (e.g. Electron) is **legitimately** installed somewhere
2. Detection rule fires
3. ALERT

Extremely Important (I)



1. Program that uses Javascript but is not listed is legitimately installed somewhere
2. Detection rule fires
3. ALERT

- ❑ Alerts must be **triaged** to **exclude false positives**

- ❑ **Huge** practical problem

- ❑ Alert triage must be based on **contextual** information

- ❑ You **cannot outsource** knowledge of your specific context

Extremely Important (II)



1. Program that uses Javascript but is not listed is legitimately installed somewhere
2. Detection rule fires
3. ALERT

- ❑ Even if your detection rules were carefully tuned...
- ❑ The IT profile of organizations **changes routinely**
 - ❑ Programs are routinely installed / modified
 - ❑ New users are routinely onboarded
 - ❑ User behavior may change
 - ❑ ...

Hmmm...

❑ `/* self-injection */`

process allocates memory page with WX access
rights AND **process.name is not (...)**

❑ Could an Attacker **modify the code** of a process listed in the rule?

❑ That would allow the Attacker to achieve self-injection without triggering the rule



Bypass: Process Hollowing

- ❑ FD = Any executable in the detection rule (easy to guess)
- ❑ Write malicious program M2 that, **during execution**:
 1. Create process P1 that executes FD in **suspended** state
 2. **Unmap** memory image of P1
 3. **Map** memory image of P1 to **M1** (the one seen before)
 4. Resume execution of P1
- ❑ P1.name = FD \Rightarrow Detection rule **not** triggered
- ❑ Malicious behavior **not** detected



Hypothesis:

Process Hollowing detected

❑ `/* self-injection */`

process allocates memory page with WX access
rights AND process.name is not (...)

❑ `/* process hollowing */`

...

❑ What you **inspected** at load time <> what you **execute**

Detection of this **suspicious**
behavior is **solved!**



Bypass: Process Doppelganging

1. Prepare a file F with content=**GoodExec**
 2. **TransactionStart** with F
 3. F.content := **BadExec**
 4. Create MemorySection **M** with F
 5. Create Process P associated with **M** (multistep legacy API)
 6. **TransactionRollback**
 7. Complete creation of P and start execution
- ❑ Existing rules do not detect this behavior
 - ❑ Scanners do **not** see **BadExec**



To make a long story short

- ❑ A **huge** amount of possibly malicious behaviors
- ❑ **Endless game**
 - ❑ Defenders detect "almost everything"
 - ❑ Attackers **discover a new trick** and **go undetected** for some time
 - ❑ Defenders **detect the new trick**
 - ❑ Attackers **discover yet another trick** and **go undetected** for some time
 - ❑ ...
- ❑ Devising / Detecting malicious behaviors requires a **deep technical knowledge**

Malware Detection: Forensics



Malware Detection: Forensics



- ❑ **Is this system / device clean or infected?**

- ❑ If infected:
 - ❑ Which malware?
 - ❑ How initial access / persistence?
 - ❑ How to restore it?

Rule of thumb for "sophisticated" malware



- ❑ IF you **don't** know whether it is infected
- ❑ THEN detection is **very hard / hardly possible**
 // just too many things to analyze

- ❑ IF you **do** know there is some malware
- ❑ THEN detection is more likely

Attack Detection: SIEM



What can be analyzed: EDR Sensors (REMINO)



1. System call invocations

2. Predefined events created by O.S. or applications

- ☐ Activity related to processes, memory, files, network

- ☐ Each sensor output ("event") has a lot of information:
 - ☐ Process ID, parent process ID, process image, ...
 - ☐ File / Message content, ...
 - ☐ ...

Fact



- ❑ System call invocations
- ❑ Predefined events created by O.S. or applications

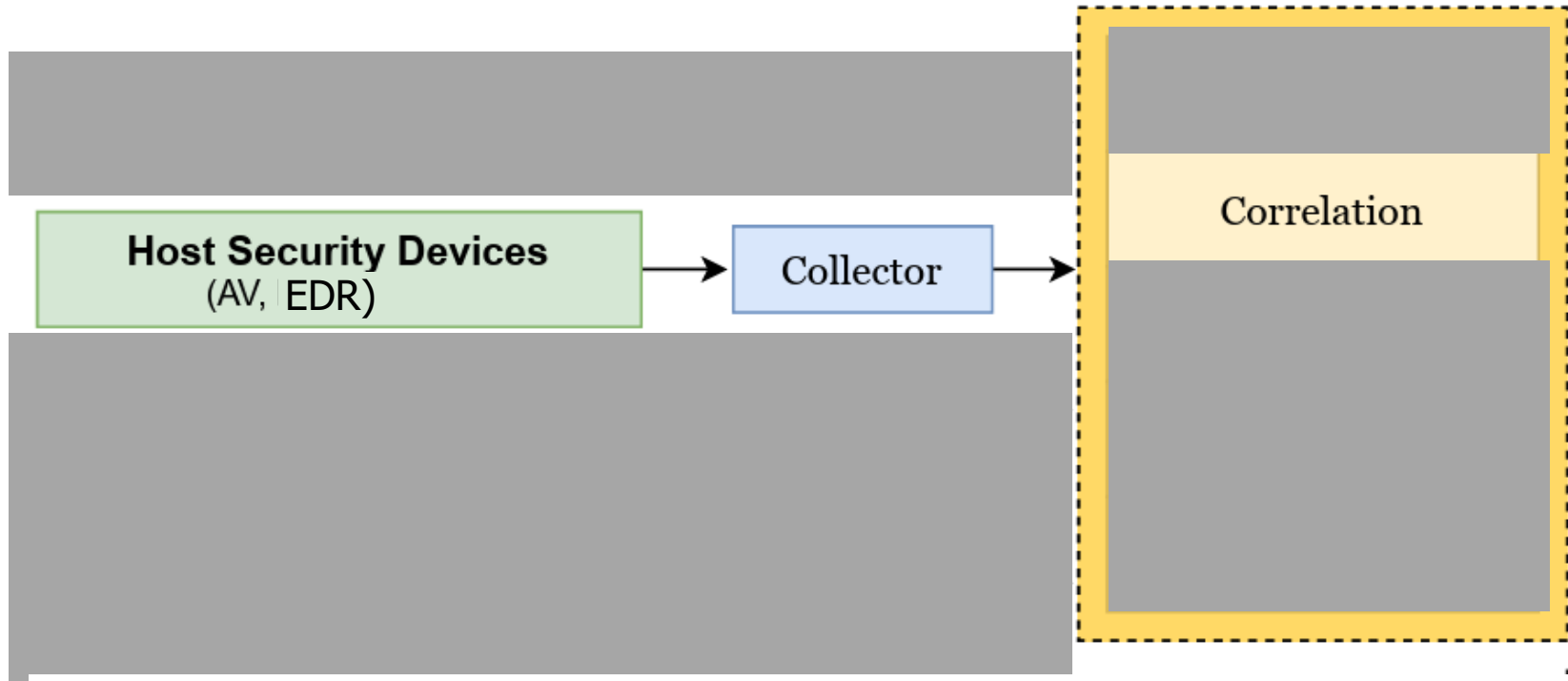
- ❑ Sensors do **not** collect **all** potentially relevant information
 1. Intrinsic technological limitations
 - ❑ Most EDRs do not log `ReqQueryMultiple`
 - ❑ August 2025: Someone has realized it can be used for exfiltration
 2. Excessive amount of information
(limited by installation-specific **configuration**)

EDR at work



1. Collect a **configuration-specific subset** of the "events" that could be collected
 2. Apply a set of **detection rules** locally
 3. Stream sensor outputs to **central platform** (perhaps with some aggregation)
-
- ❑ Central platform will apply detection rules on sensor outputs from **multiple endpoints**

Attack detection: SIEM

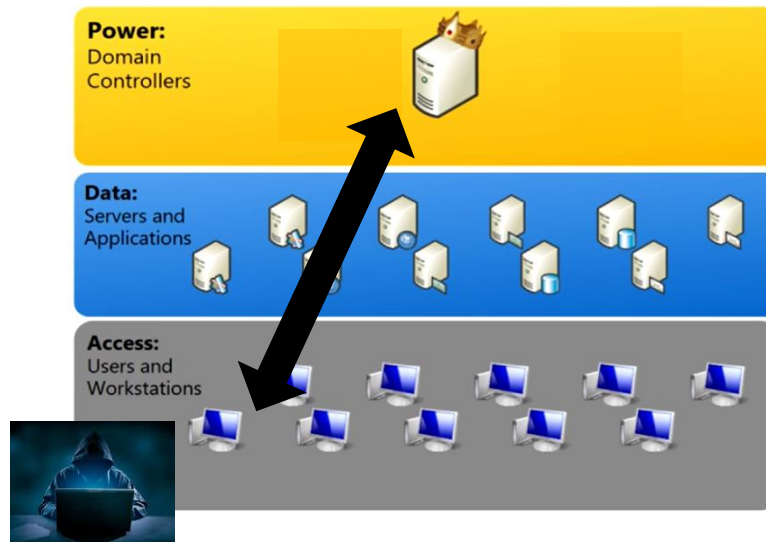


- ❑ Central platform will apply detection rules on sensor outputs from **multiple endpoints**

Example:

Kerberoasting Attack

1. Connect with Domain Controller on port 88
2. Kerberos protocol:
 - a. Authenticate as user U (credentials needed)
 - b. Ask **service ticket ST(U,S)** for some service S
3. Attempt to crack **ST(U,S)** for **finding password** of service account S



How to detect?



Example Detection: Tool



❑ Sensors collect:

- ❑ ...

- ❑ Process creation for executing a file

- ❑ Command line arguments

- ❑ ...

❑ Detection rule:

- ❑ Execution of Bifrost software tool with certain invocation arguments

- ❑ Consider invocation arguments because changing filename is very easy

```
event.category:process and event.type:start and process.args:  
("-action" and ("-kerberoast" or askhash or asktgs or asktgt or  
s4u or ("-ticket" and ptt) or (dump and (tickets or keytab))))
```

Hmmm...



- ❑ Detection rule:

- ❑ Execution of Bifrost software tool with certain invocation arguments
 - ❑ Consider invocation arguments because changing filename is very easy

- ❑ Attacker may:

1. Modify Bifrost software tool so that invocation arguments are different
2. Recompile it

or

1. Use a different software tool (e.g., Rubeus)

- ❑ **Detection rule not triggered!**

Example Detection: Anomalous Network Comm.

❑ Sensors collect:

- ❑ ...

- ❑ TCP connections

- ❑ Executable file of the client process that creates the connection

- ❑ ...

❑ Detection rule:

- ❑ Connection to port 88 by **unusual clients**

```
network where event.type == "start" and network.direction ==  
"outgoing" and destination.port == 88 and source.port >= 49152  
and process.name not in ("swi_fc.exe", "fsIPcam.exe",  
"IPCamera.exe", "MicrosoftEdgeCP.exe", "MicrosoftEdge.exe",  
"iexplore.exe", "chrome.exe", "msedge.exe", "\"opera.exe",  
"firefox.exe")
```


Detection Rules: Fragile vs Robust



- Detection rules may be:

- Fragile:

- **Easy** to write and test

- (and to circumvent)

- **Cheap** to Defenders

- (and to Attackers)

- Robust

- **Difficult** to write and test

- (and to circumvent)

- **Costly** to Defenders

- (and to Attackers)

- In practice you have a mix of both

Scores



- ❑ Alerts are hardly the result of a **single** detection rule
- ❑ Common heuristics:
 - ❑ Each detection rule has a predefined **score**
 - ❑ When an Account exceeds a predefined **aggregate score** in a predefined **time interval**
→ Alert

False Positives

❑ Detection rule:

❑ Connection to port 88 by unusual clients

```
...and process.name not in ("swi_fc.exe", "fsIPcam.exe",  
"IPCamera.exe", "MicrosoftEdgeCP.exe", "MicrosoftEdge.exe",  
"iexplore.exe", "chrome.exe", "msedge.exe", "\"opera.exe",  
"firefox.exe")
```

❑ What if some new software that **legitimately** asks for service tickets is installed?

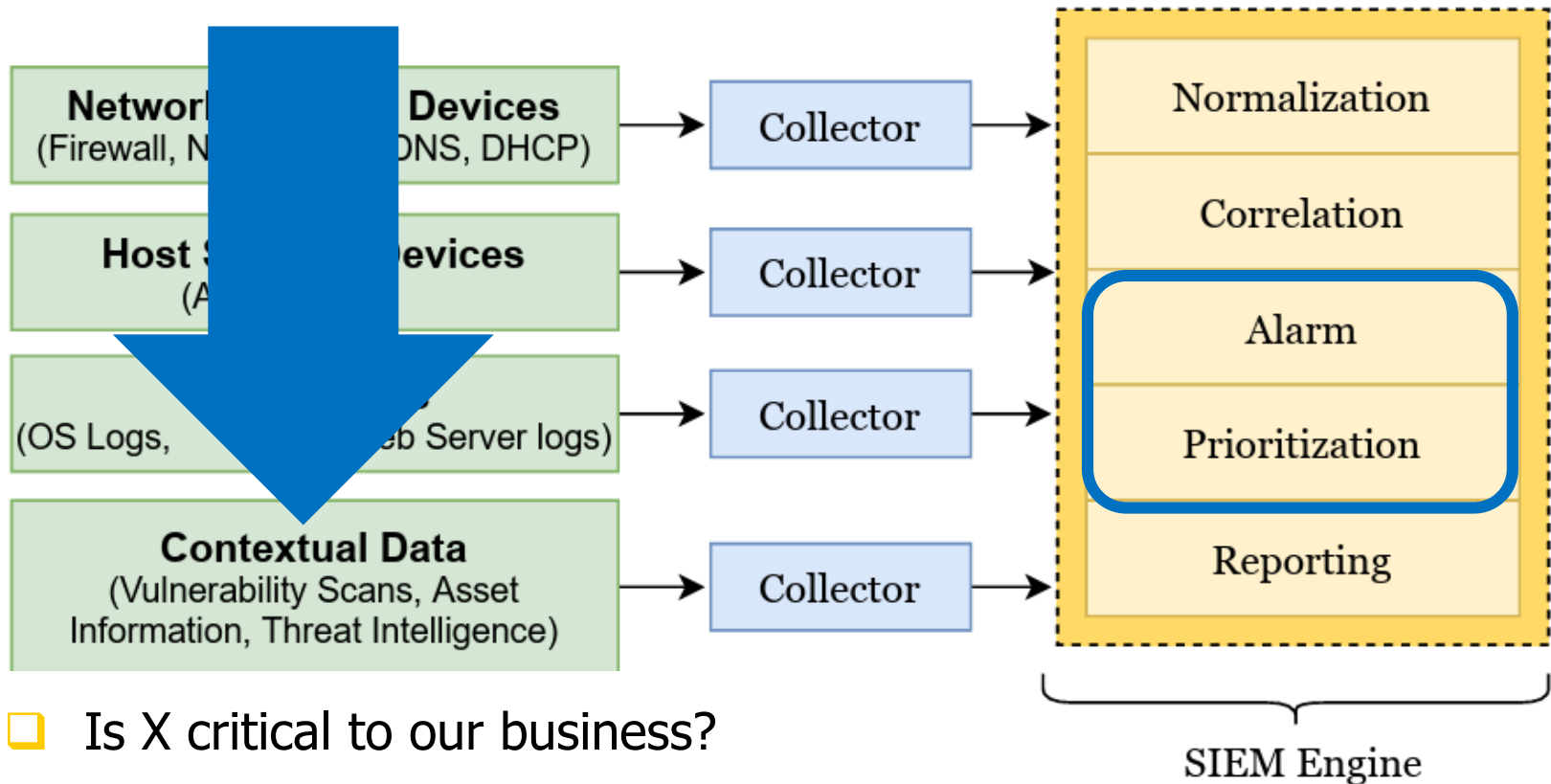
Extremely Important (REMIND)



1. Program that uses ... legitimately installed somewhere
2. Detection rule fires
3. ALERT

- ❑ Alerts must be triaged to **exclude false positives**
 - ❑ **Huge** practical problem
- ❑ Alert triage must be based on **contextual** information
 - ❑ You **cannot outsource** knowledge of your specific context
- ❑ Even if your detection rules were carefully tuned
 - ...The IT profile of organizations **changes routinely**

Contextual Information



- ☐ Is X critical to our business?
- ☐ Does Y work from abroad?
- ☐ ...

Kerberoasting:

Other Detection Logic (I)



- ❑ Attempt to crack $ST(U,S)$ for **finding password** of service account S
- ❑ Attacker does not know in advance whether cracking will succeed: usually **several** service tickets are requested: $ST(U,S1)$, $ST(U,S2)$, $ST(U,S3)$, ...
- ❑ Detection rule:
 - ❑ Is there any user that asked too many ST in a predefined amount of time?

Kerberoasting:

Other Detection Logic (II)



- ❑ One does not know in advance whether cracking will succeed
- ❑ Attackers often ask for **several** service tickets:
ST(U,S1), ST(U,S2), ST(U,S3), ...
- ❑ Attackers often ask for **certain service names**: those that were likely installed many years ago (have weak configuration)
- ❑ Detection rule(s):
 - ❑ Is there any user that asked ST for service names that **do not exist**?
 - ❑ ...and that asked for several ST in a short time?

Detection Rules: Execution Cost



- ❑ Certain detection rules require **searching for / counting multiple events** in a specified **time interval**
- ❑ There are usually **millions** of events to consider
- ❑ Detection rules qualities:
 - ❑ Fragile / Robust
 - ❑ Precision
 - ❑ **Execution cost**

Detection Rules:

Practical Remark



- ❑ **Many detection rules can be bypassed**
 - ❑ Just think a little about the previous examples
- ❑ It is mostly a matter of **Attacker effort**
- ❑ Knowledge of the detection rule helps a lot

Evasion

A thick, horizontal yellow brushstroke that spans most of the width of the slide, positioned directly below the title 'Evasion'.

Lack of detection



- ❑ Attacker activity may go undetected because:
 1. **Sensor** do not collect relevant information
 2. **Detection rules** do not trigger
 3. **Alerts** are **not investigated** or they are deemed **false positives**

- ❑ **Evasion** (bypass) refers to activity that aims at 1 / 2

Evasion (Bypass) (I)



☐ Perceptual

- ☐ Relevant information **cannot be collected** due to technological limitation

☐ Configuration

- ☐ Relevant information **is not collected** due to configuration

☐ Related to **sensors**

Evasion (Bypass) (II)



❑ Logical

- ❑ Relevant information has been collected but detection rules have a logical gap
- ❑ No detection rule for the specific activity

❑ Classification

- ❑ Detection rules are volumetric or score-based
- ❑ Relevant information in the considered time interval is not enough to trigger the rule

❑ Related to **detection rules**

Practical Remark



- ❑ Attacker activity may go undetected because:
 1. **Sensor** do not collect relevant information
 2. **Detection rules** do not trigger
 3. **Alerts** are **not investigated** or they are deemed **false positives**

- ❑ **In most cases, THIS is THE problem**

Ahem...



Advisory on New Endpoint Detection and Response (EDR) Killer Tool Used by Multiple Ransomware Groups

16 August 2025

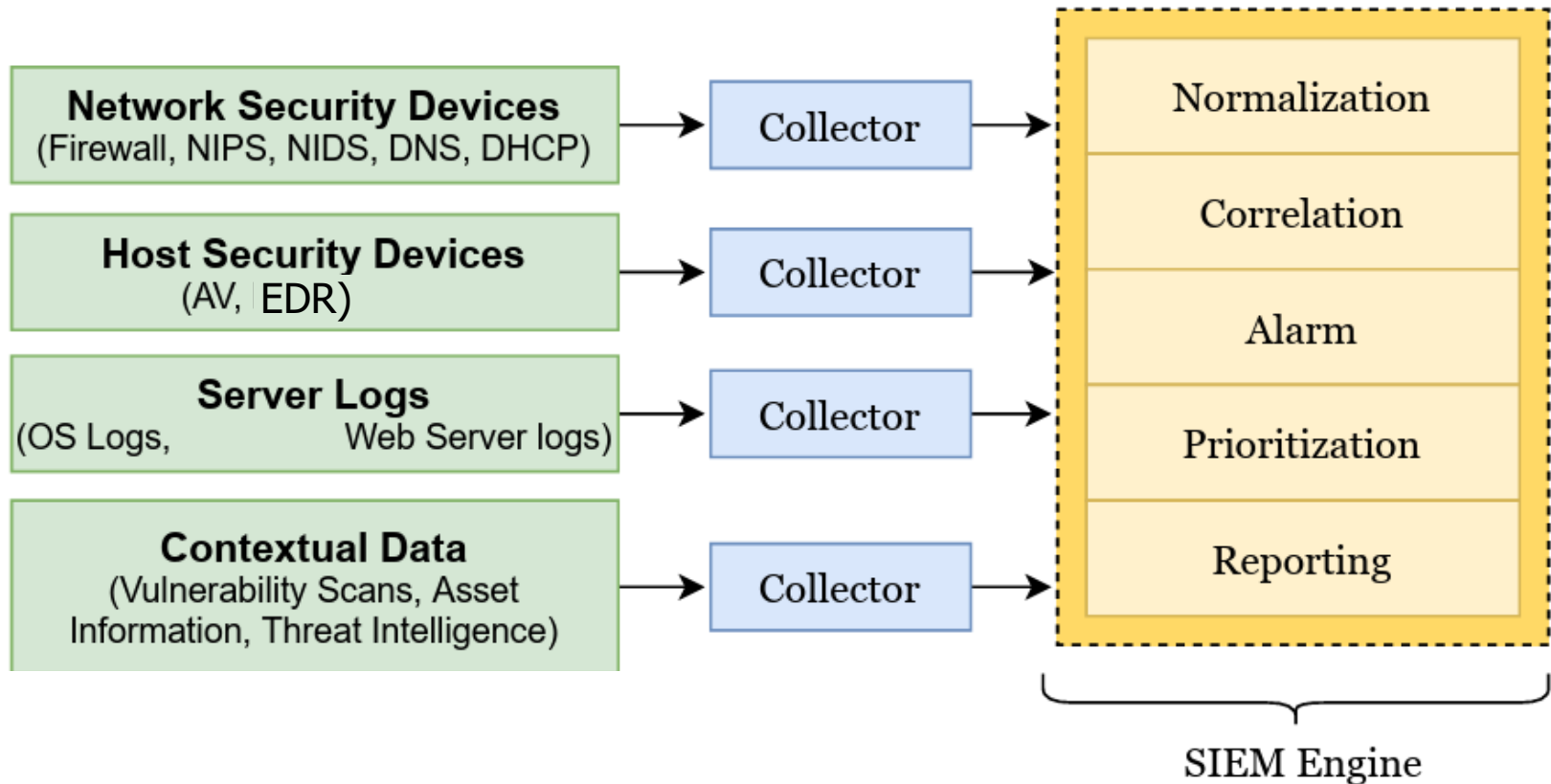
There have been reports of a new malicious tool, known as Endpoint Detection and Response (EDR) killer, being actively used by at least eight ransomware groups to disable EDR solutions.

Cyber Security Agency of Singapore

SIEM in practice

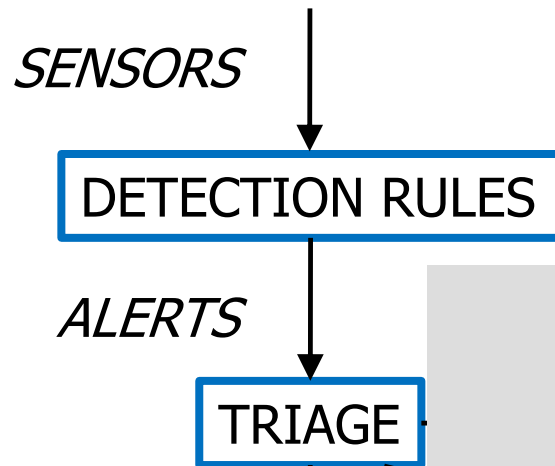


SIEM: MANY sensors

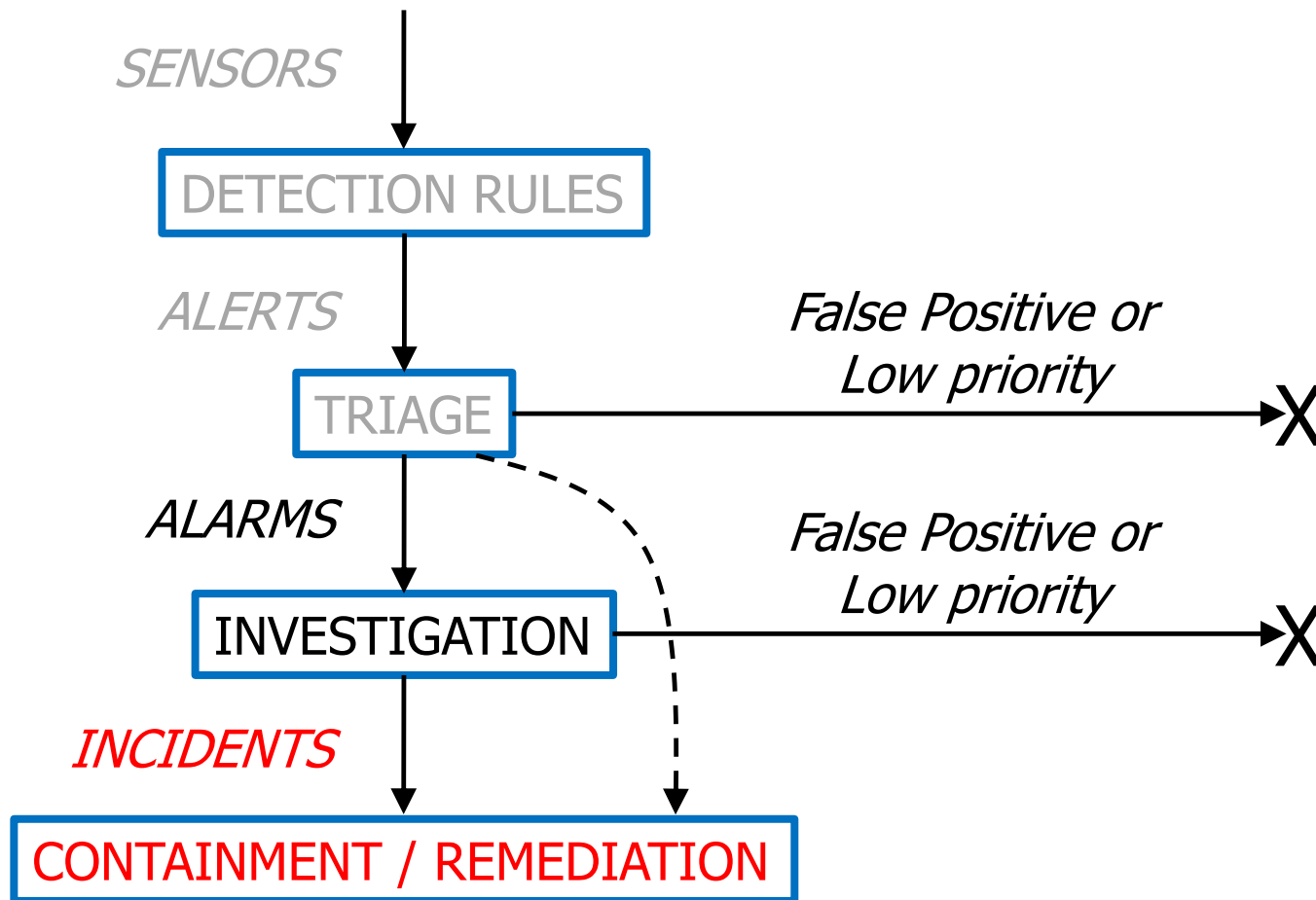


99% False Positives: A Qualitative Study of SOC Analysts' Perspectives on Security Alarms
31st USENIX Symposium

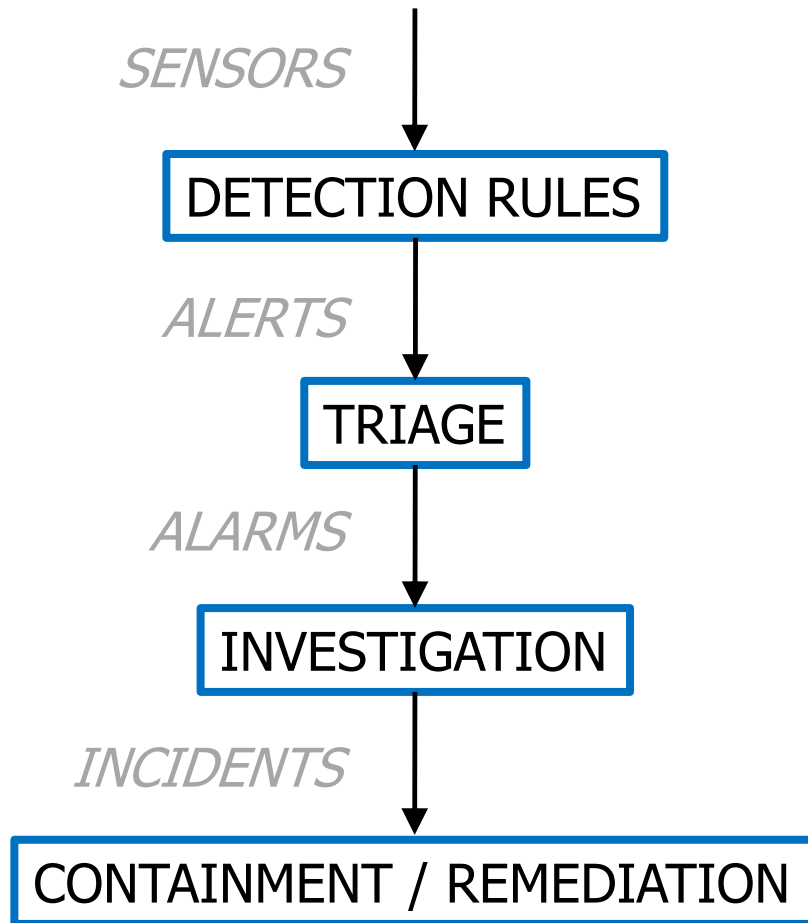
SIEM: Logical workflow (I-a)



SIEM: Logical workflow (I-b)



SIEM: Logical workflow (II)



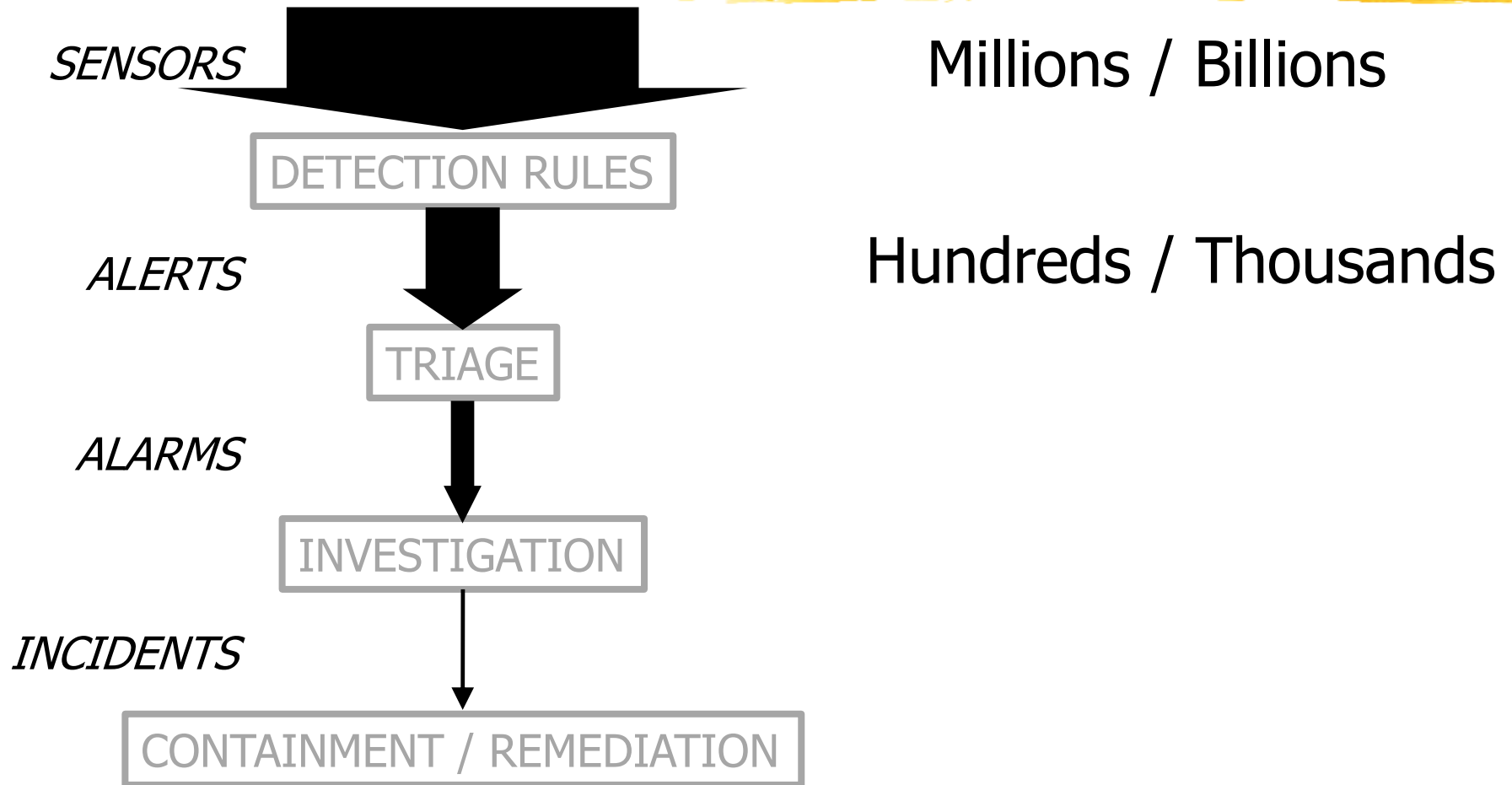
- ☐ **Automatic**
- ☐ Detection Engineers

- ☐ **Partly manual**
- ☐ SOC Analyst

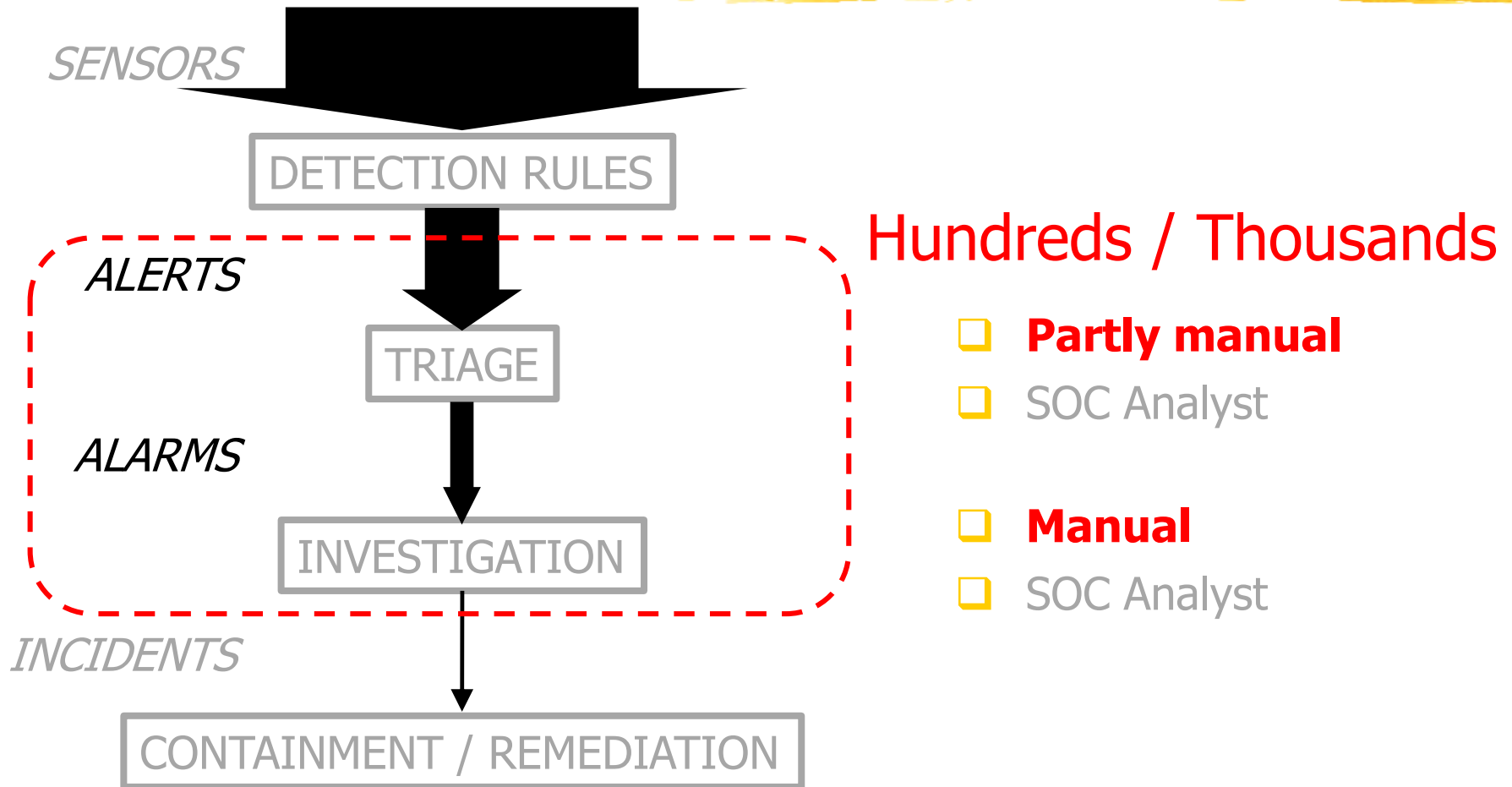
- ☐ **Manual**
- ☐ SOC Analyst

- ☐ **Manual**
- ☐ Incident Responder

Daily Volumes (Typical)



"THE" Practical Problem



Just one of the many studies (I)

Global Security Operations Center Study Results

MORNING CONSULT

IBM

MARCH 2023

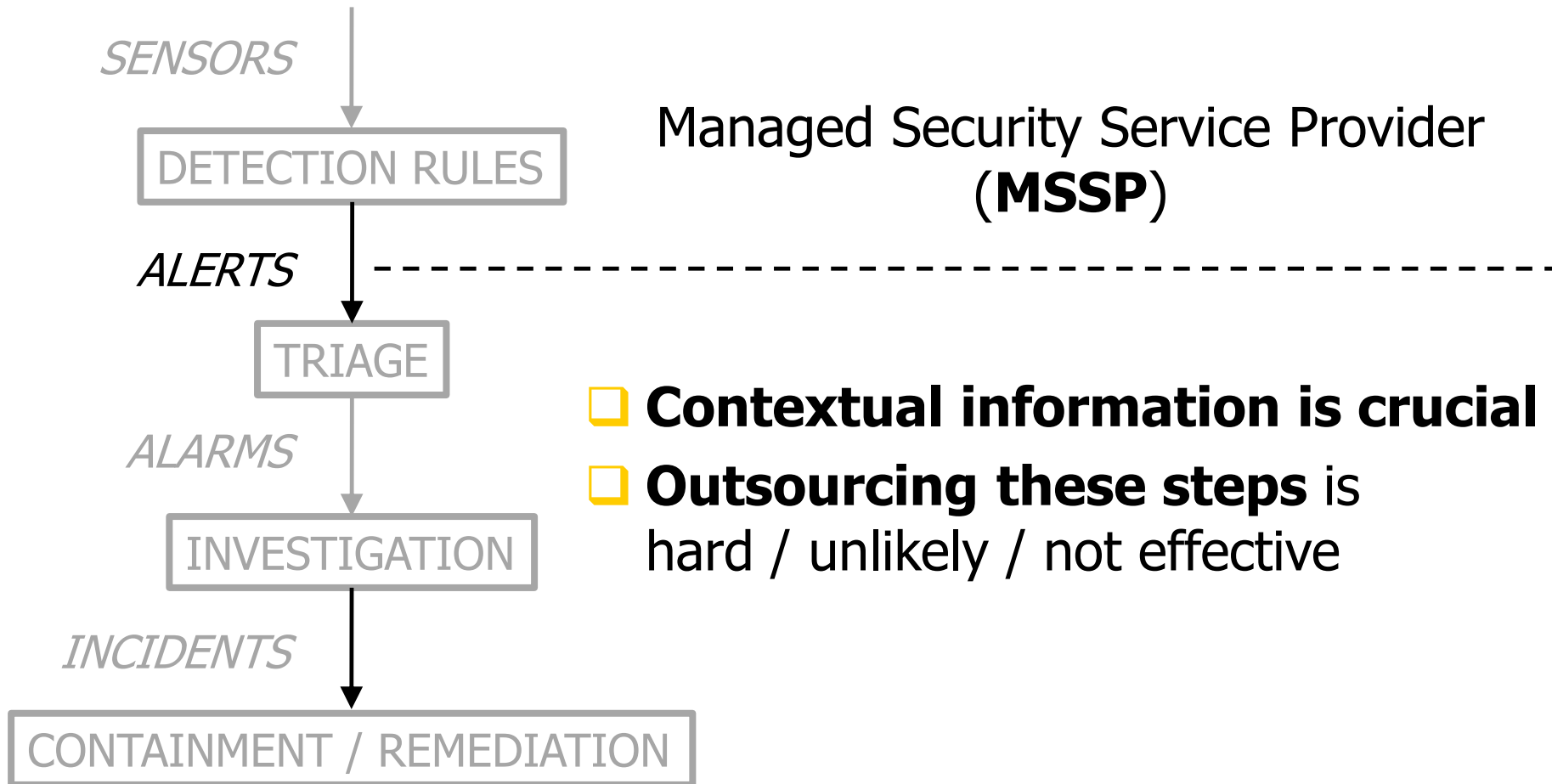
- ❑ 1000 SOC members, 100 orgs, 10 countries
- ❑ Orgs with >1000 employees
- ❑ Certainly high budget and awareness

Just one of the many studies (II)



- ❑ *We can investigate **less than half** of all alerts*
- ❑ ***Most** are either **false positives** or **low priority***
- ❑ *Over the last two years, the time it takes to investigate an alert has **increased***

Outsourcing?



To make a long story short



- ❑ The problem is **not** "detecting attacks"
- ❑ The problem is "detecting **only** attacks" (i.e., with low false positives)

- ❑ AI? Good luck
 - ❑ **Contextual** information?
 - ❑ **Explanation** of alerts?