The Web Application Strikes Back

JavaOne 2015

Dominik Schadow | bridgingIT

Application Intrusion Detection with OWASP AppSensor

Agenda

Basics

Architect

Implement

Basics

Attacks are often executed as trial & error

Most web applications don’t inform about ongoing attacks

You learn about successful attack after the data is leaked

Automatically react on identified attacks in realtime

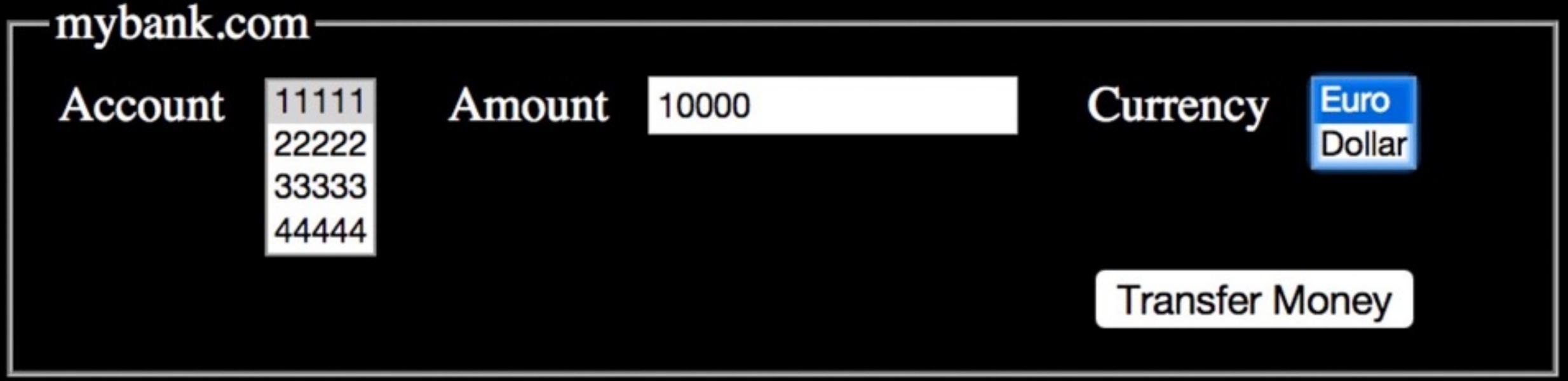
Stop attackers before they are successful

It’s impossible to detect every attack

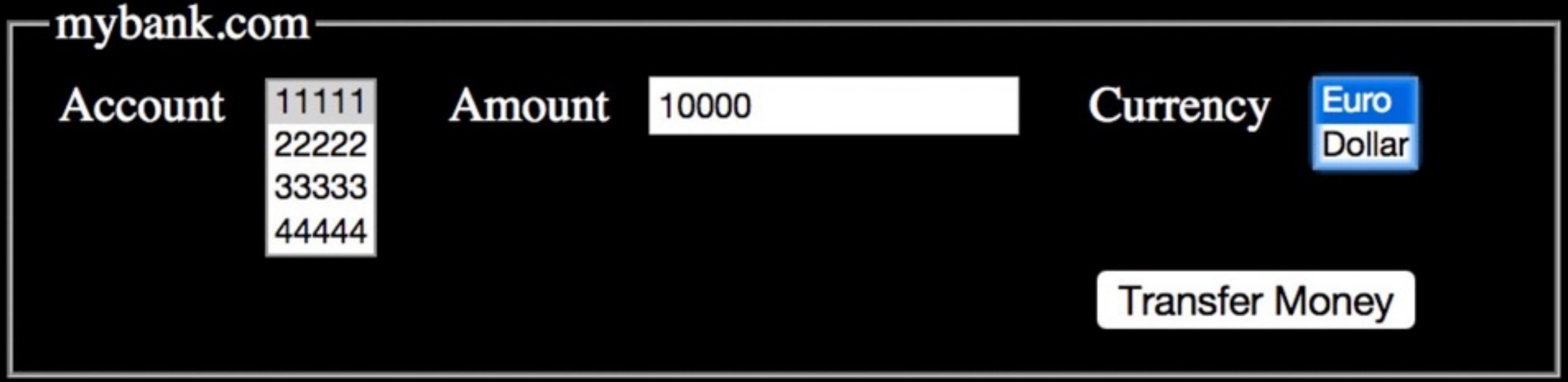
But enough to know about a user’s intentions

What makes application intrusion detection more effective than a web application firewall?

?accountNo=66666&amount=10000&currency=Euro



?accountNo=66666&amount=10000&currency=Euro



?accountNo=66666&amount=10000&currency=Euro

Context matters

Application is aware of valid and invalid data

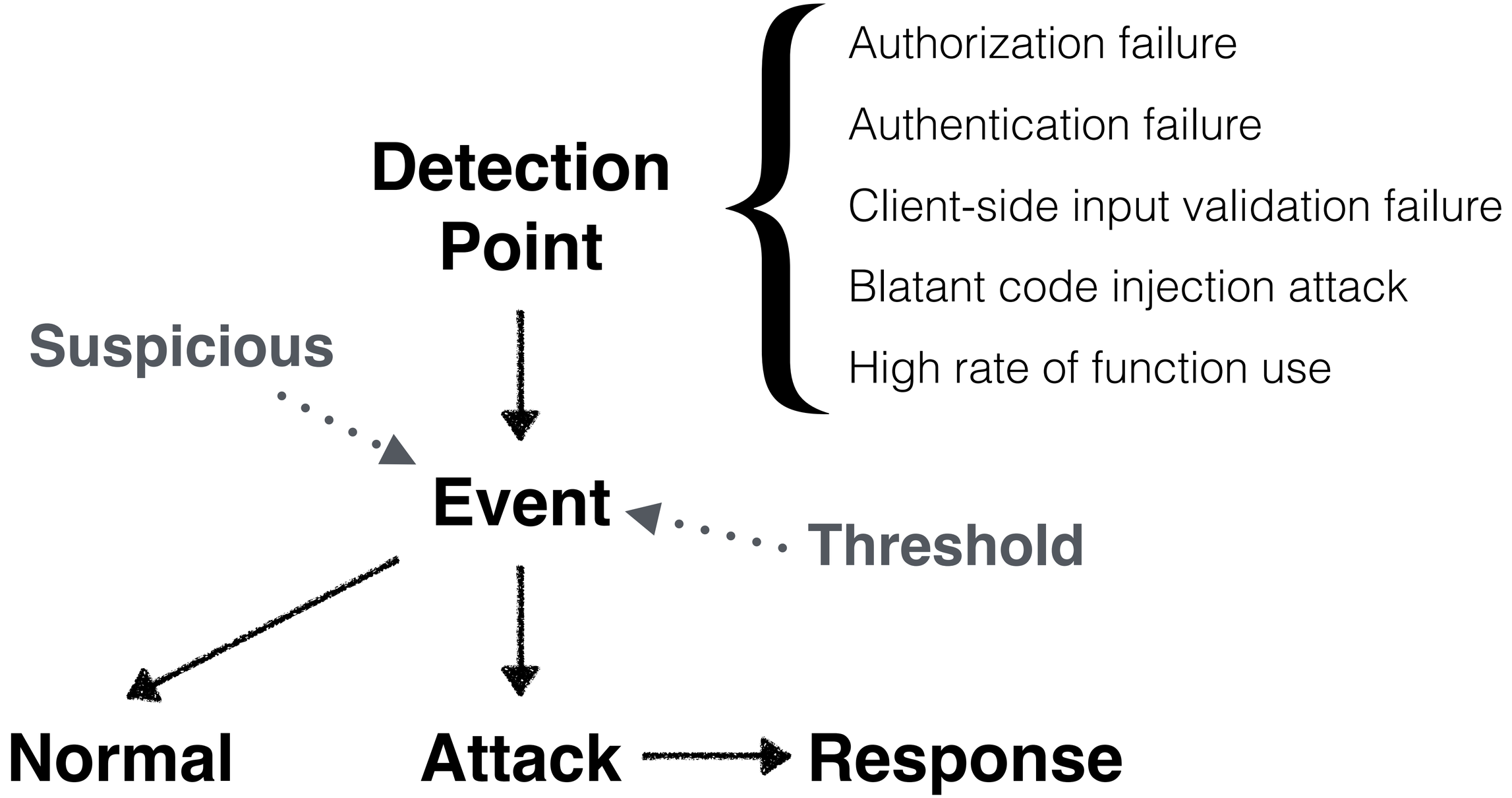
Application is aware of the business context

Application aware defense

Part of a defense in depth strategy

Detect attacks, not vulnerabilities

Application has to be secure


Detection Point
Authorization failure
Authentication failure
Client-side input validation failure
Blatant code injection attack
High rate of function use
Suspicious
Event
Threshold
Normal
Attack
Response

Detection Points

Detect events in the application (like a sensor)

Tightly integrated in the application code
(or outside e.g. in a WAF)

Useable in any backend application layer

```
@Named
public class EncounterValidator implements Validator {
 @Autowired
 private SpringValidatorAdapter validator;

 public void validate(Object target, Errors errors) {
  validator.validate(target, errors);
  // ...
 }
}
```

```
@Named
public class EncounterValidator implements Validator {
 @Autowired
 private SpringValidatorAdapter validator;

 public void validate(Object target, Errors errors) {
  validator.validate(target, errors);
  // ...
 }
}
```

```
@Autowired
private AppSensorClient appSensorClient;
@Autowired
private EventManager ids;

public void validate(Object target, Errors errors) {
 validator.validate(target, errors);
 Encounter encounter = (Encounter) target;

 if (hasXssPayload(encounter.getEvent())) {
  fireXssEvent();
  errors.rejectValue("event", "xss.attempt", "Stop!");
 }
}
```

```
@Autowired
private AppSensorClient appSensorClient;
@Autowired
private EventManager ids;

public void validate(Object target, Errors errors) {
 validator.validate(target, errors);
 Encounter encounter = (Encounter) target;

 if (hasXssPayload(encounter.getEvent())) {
  fireXssEvent();
  errors.rejectValue("event", "xss.attempt", "Stop!");
 }
}
```

May use **blacklists** to identify attacks

```
private boolean hasXssPayload(String text) {
 return StringUtils.containsIgnoreCase(text, "<")
  || StringUtils.containsIgnoreCase(text, "javascript");
}
```

```
private boolean hasXssPayload(String text) {
 return StringUtils.containsIgnoreCase(text, "<")
  || StringUtils.containsIgnoreCase(text, "javascript");
}
```

Often already used in isolated application parts

Improve to central analysis and response unit

```
@Autowired
private AppSensorClient appSensorClient;
@Autowired
private EventManager ids;

public void validate(Object target, Errors errors) {
 validator.validate(target, errors);
 Encounter encounter = (Encounter) target;

 if (hasXssPayload(encounter.getEvent())) {
  fireXssEvent();
  errors.rejectValue("event", "xss.attempt", "Stop!");
 }
}
```

Detection Point Categories

Request Exception (RE)
Authentication Exception (AE)
Session Exception (SE)
Access Control Exception (ACE)
Input Exception (IE)
Encoding Exception (EE)
Command Injection Exception (CIE)
FileIO Exception (FIO)
User Trend Exception (UT)
System Trend Exception (STE)

Input Exception Detection Points

IE1 - Cross-Site Scripting attempt

IE2 - violation of implemented white lists

IE3 - violation of implemented black lists

IE4 - violation of input data integrity

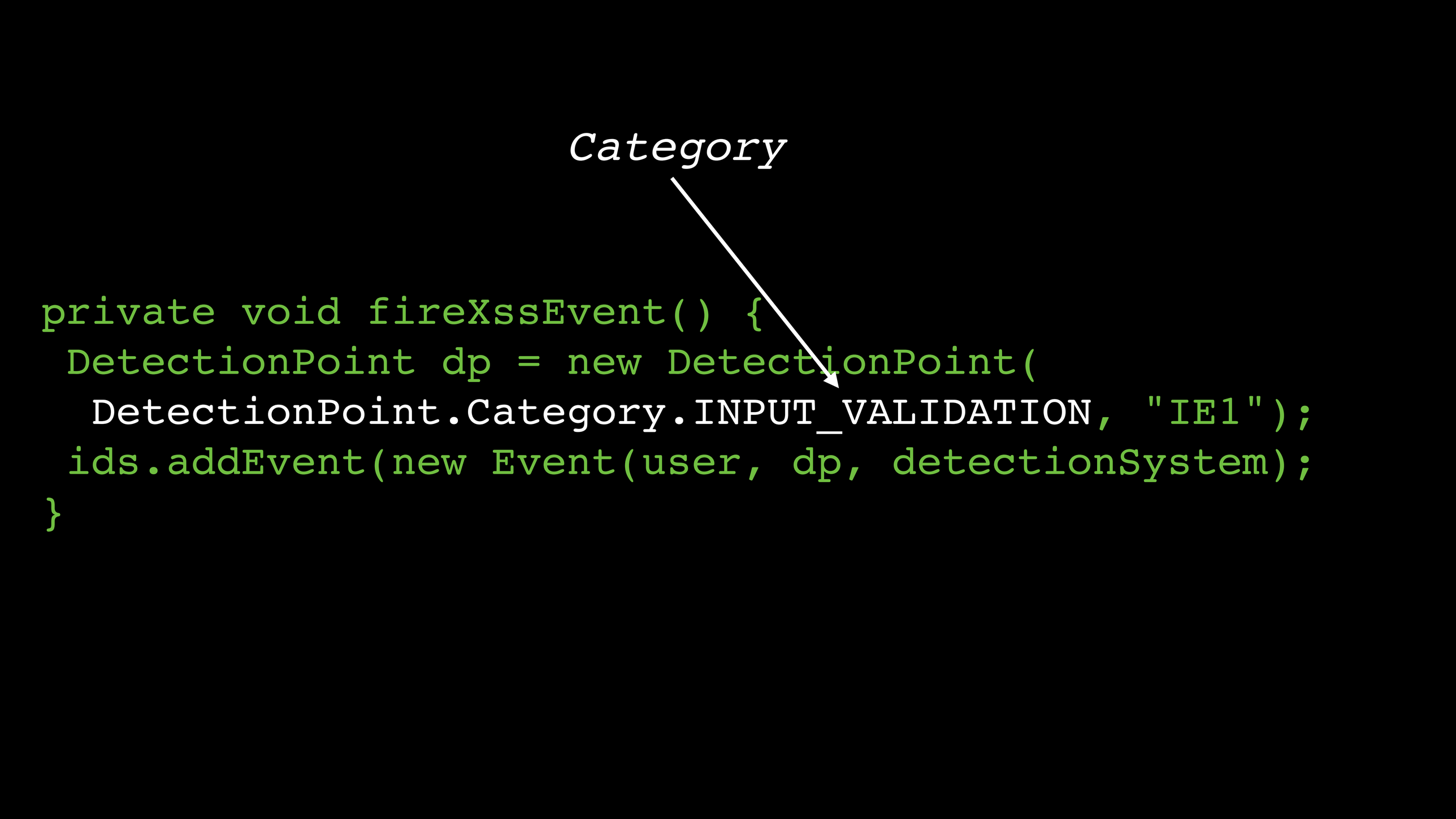
IE5 - violation of stored business data integrity

IE6 - violation of security log integrity

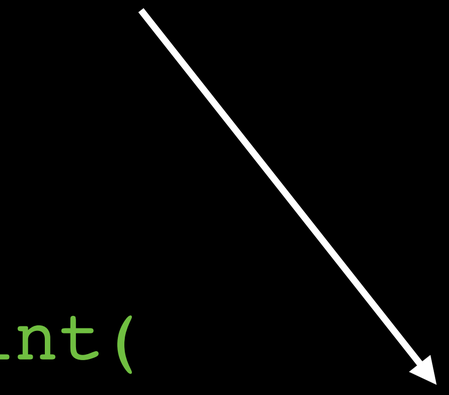
IE7 - detect abnormal content output structure

```
private void fireXssEvent() {
 DetectionPoint dp = new DetectionPoint(
  DetectionPoint.Category.INPUT_VALIDATION, "IE1");
 ids.addEvent(new Event(user, dp, detectionSystem);
}
```

```
private void fireXssEvent() {
 DetectionPoint dp = new DetectionPoint(
  DetectionPoint.Category.INPUT_VALIDATION, "IE1");
 ids.addEvent(new Event(user, dp, detectionSystem);
}
```

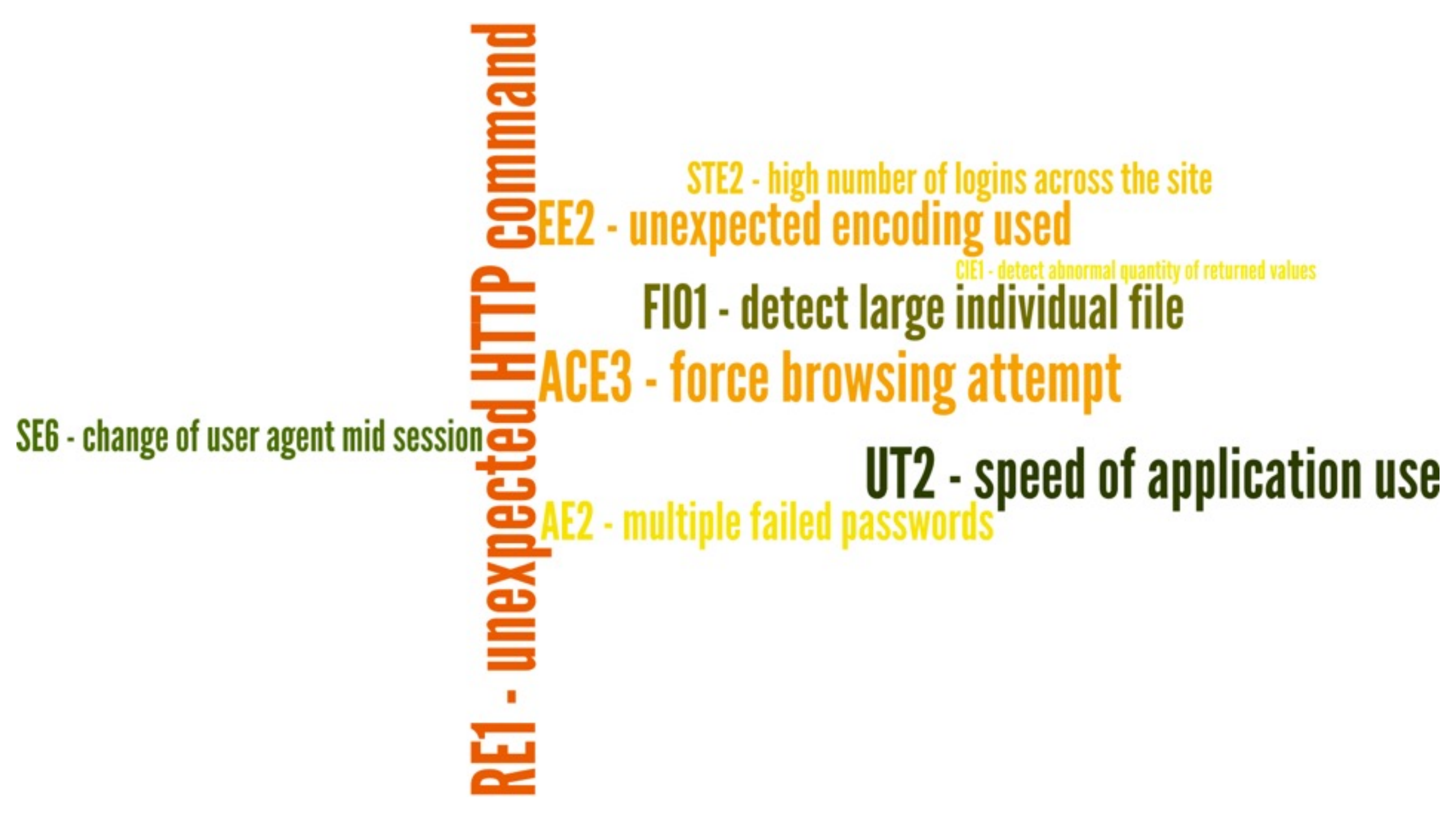

Category
private void fireXssEvent() {
DetectionPoint dp = new DetectionPoint(
DetectionPoint.Category.INPUT_VALIDATION, "IE1");
ids.addEvent(new Event(user, dp, detectionSystem);
}

Label



```
private void fireXssEvent() {
 DetectionPoint dp = new DetectionPoint(
  DetectionPoint.Category.INPUT_VALIDATION, "IE1");
 ids.addEvent(new Event(user, dp, detectionSystem);
}
```

IE1-001, IE1-002, …


RE1 - unexpected HTTP command
STE2 - high number of logins across the site
EE2 - unexpected encoding used
CIE1 - detect abnormal quantity of returned values
FIO1 - detect large individual file
ACE3 - force browsing attempt
SE6 - change of user agent mid session
UT2 - speed of application use
AE2 - multiple failed passwords

Events

Observable occurrences in an application

Monitored and analyzed to determine an attack

```
private void fireXssEvent() {
 DetectionPoint dp = new DetectionPoint(
  DetectionPoint.Category.INPUT_VALIDATION, "IE1");
 ids.addEvent(new Event(user, dp, detectionSystem);
}
```

```
@Autowired
private AppSensorClient appSensorClient;
@Autowired
private EventManager ids;

public void validate(Object target, Errors errors) {
 validator.validate(target, errors);
 Encounter encounter = (Encounter) target;

 if (hasXssPayload(encounter.getEvent())) {
  fireXssEvent();
  errors.rejectValue("event", "xss.attempt", "Stop!");
 }
}
```

Thresholds

React immediately to malicious events (**threshold = 1**)

Requires **count**

React delayed to suspicious events (**threshold > 1**)

Requires **count** and **interval**

Threshold Examples

5 events over the last 4 hours

Any 10 events over the last 24 hours

200% increase over one hour

System Trends

More difficult to configure

Require a large user base/ frequent usage

Consider special situations (sale, holidays, …)

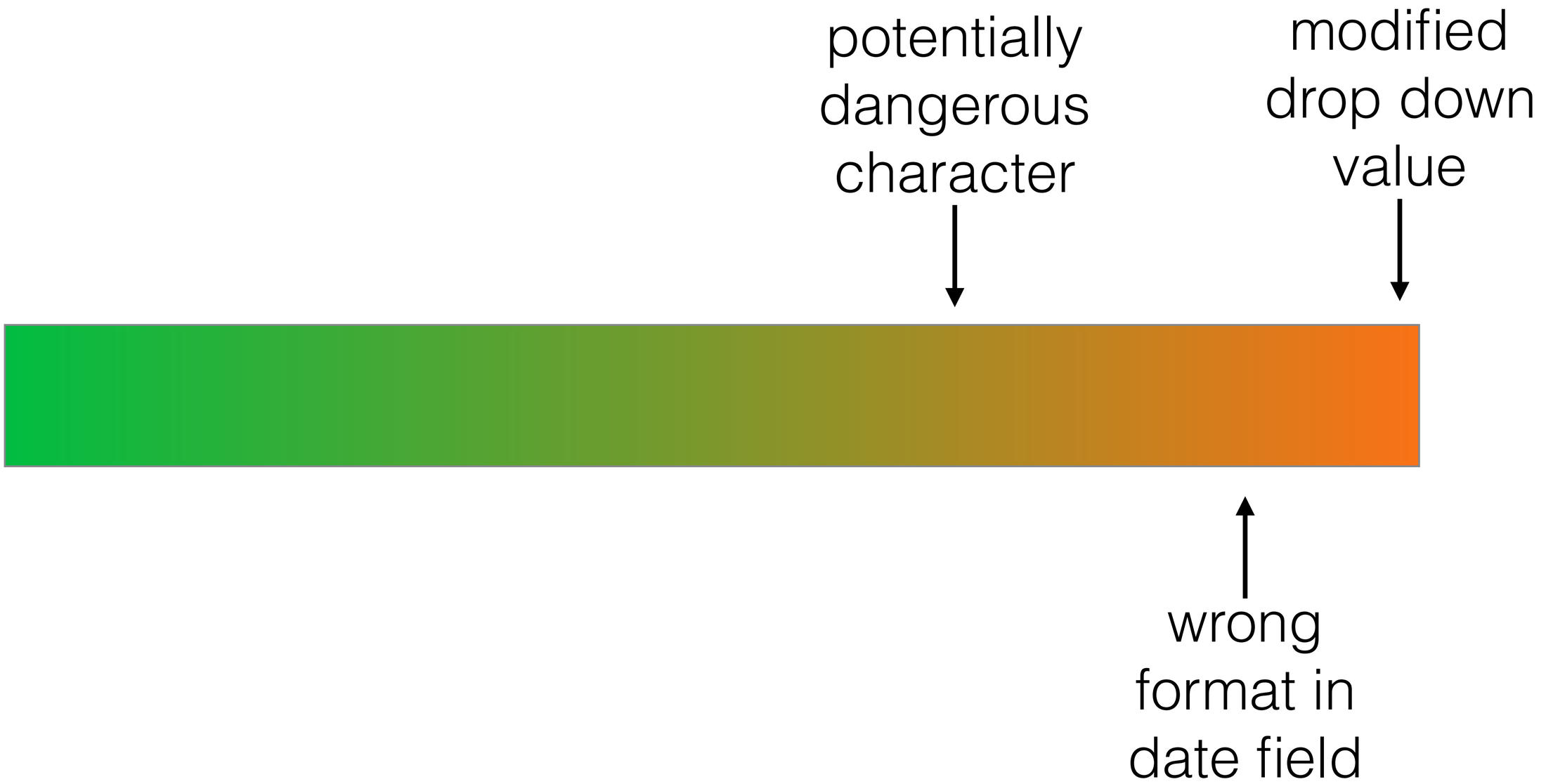
Without client-side validation

wrong format in date field

potentially dangerous character

modified drop down value

With client-side validation



Responses

Defend the application, its data and its users

Must happen automatically in realtime

Respond to an attack in any way you like

No retaliation


Logging change
No response
User status change
Account lockout
Timing change
Function disabled
Account logout
Collect data from user
User notification
Proxy
Admin notification
Process terminated
Application disabled
Function amended

Respond to suspicious events and identified attacks differently

Available responses are listed in the `xxxResponseHandler` class

LocalResponseHandler

log
logout
disableUser
disableComponentForSpecificUser
disableComponentForAllUsers

Responses and Thresholds

Threshold	Response
2. event	Log request
4. event	Logout user
6. event	Disable user

```
<detection-point>
  <category>Input Validation</category>
  <id>IE1</id>
  <threshold>
   <count>3</count>
   <interval unit="minutes">5</interval>
  </threshold>
  <responses>
   <response>
    <action>log</action>
   </response>
   <response>
    <action>logout</action>
   </response>
  </responses>
 </detection-point>
```

```
<detection-point>
  <category>Input Validation</category>
  <id>IE1</id>
  <threshold>
   <count>3</count>
   <interval unit="minutes">5</interval>
  </threshold>
  <responses>
   <response>
    <action>log</action>
   </response>
   <response>
    <action>logout</action>
   </response>
  </responses>
 </detection-point>
```

```
<detection-point>
  <category>Input Validation</category>
  <id>IE1</id>
  <threshold>
   <count>3</count>
   <interval unit="minutes">5</interval>
  </threshold>
  <responses>
   <response>
    <action>log</action>
   </response>
   <response>
    <action>logout</action>
   </response>
  </responses>
 </detection-point>
```

```
<detection-point>
  <category>Input Validation</category>
  <id>IE1</id>
  <threshold>
   <count>3</count>
   <interval unit="minutes">5</interval>
  </threshold>
  <responses>
   <response>
    <action>log</action>
   </response>
   <response>
    <action>logout</action>
   </response>
  </responses>
 </detection-point>
```

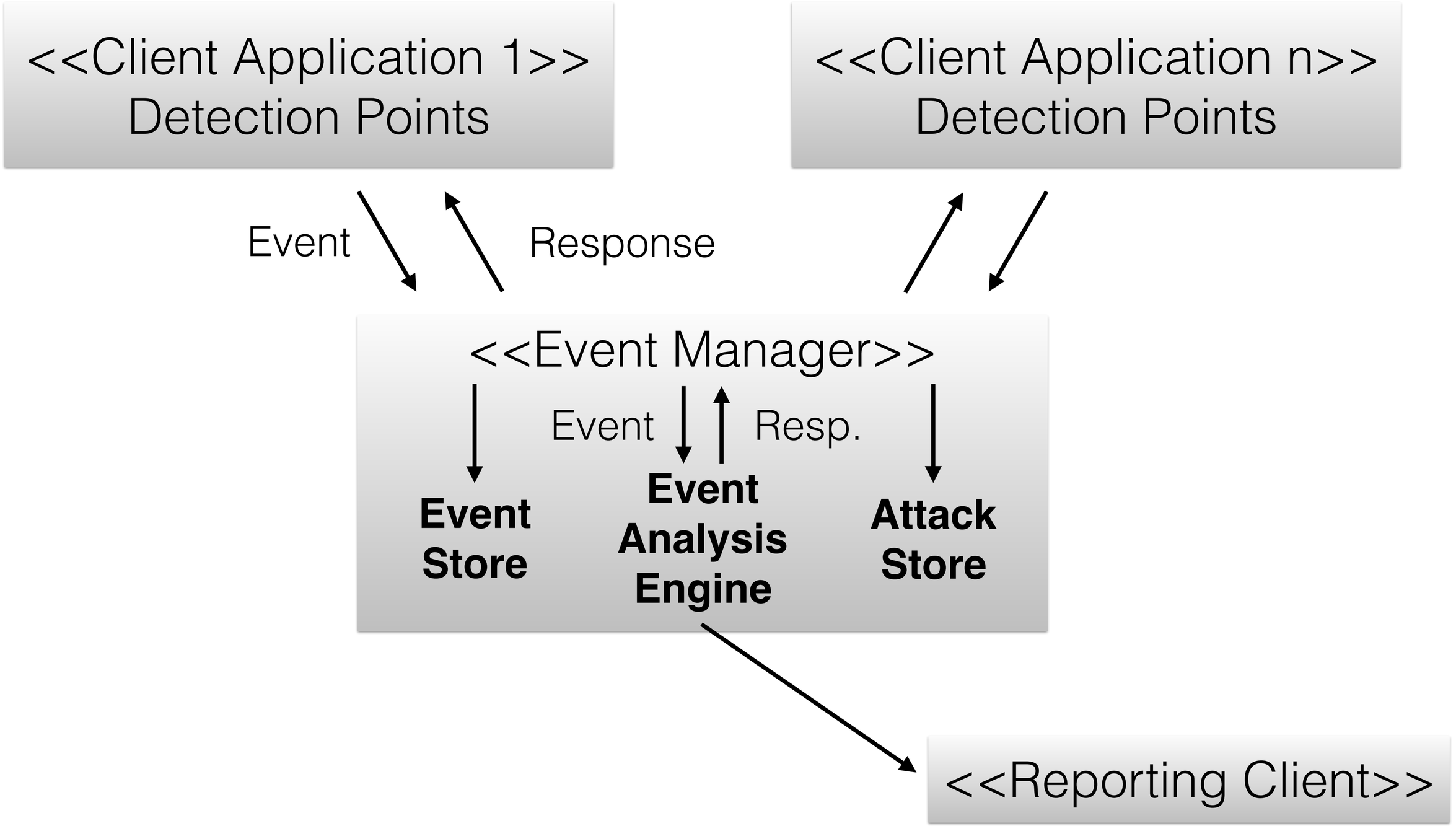
```
<detection-point>
  <category>Input Validation</category>
  <id>IE1</id>
  <threshold>
   <count>3</count>
   <interval unit="minutes">5</interval>
  </threshold>
  <responses>
   <response>
    <action>log</action>
   </response>
   <response>
    <action>logout</action>
   </response>
  </responses>
 </detection-point>
```

React on anonymous and authenticated users

```
private void fireEvent() {
 if (user.isAnonymous()) {
  DetectionPoint dp = new DetectionPoint(
   INPUT_VALIDATION, "IE1-001");
  ids.addEvent(new Event(user, dp, detectionSystem));
 } else {
  DetectionPoint dp = new DetectionPoint(
   INPUT_VALIDATION, "IE1-002");
  ids.addEvent(new Event(user, dp, detectionSystem));
 }
}
```

```
private void fireEvent() {
 if (user.isAnonymous()) {
  DetectionPoint dp = new DetectionPoint(
   INPUT_VALIDATION, "IE1-001");
  ids.addEvent(new Event(user, dp, detectionSystem));
 } else {
  DetectionPoint dp = new DetectionPoint(
   INPUT_VALIDATION, "IE1-002");
  ids.addEvent(new Event(user, dp, detectionSystem));
 }
}
```

```
private void fireEvent() {
 if (user.isAnonymous()) {
  DetectionPoint dp = new DetectionPoint(
   INPUT_VALIDATION, "IE1-001");
  ids.addEvent(new Event(user, dp, detectionSystem));
 } else {
  DetectionPoint dp = new DetectionPoint(
   INPUT_VALIDATION, "IE1-002");
  ids.addEvent(new Event(user, dp, detectionSystem));
 }
}
```


<<Client Application 1>>
Detection Points
<<Client Application n>>
Detection Points
Event
Response
<<Event Manager>>
Event
Resp.
Event
Store
Event
Analysis
Engine
Attack
Store
<<Reporting Client>>

Architect

Execution modes to access the analysis engine

	REST	SOAP	Thrift	RabbitMQ	Kafka	Local
Application language	Java & non Java					Java
# of applications	multiple					1
Cluster support	yes					no

Client and Server

Client application (detection unit)

- Determines detection points
- Signals all occurring events

Server application (response unit)

- Evaluates events against configured policy
- Generates responses

Web Services (REST)

/events
POST with JSON event data

/attacks
POST with JSON attack data

/responses
GET

Handling detection and response inside one application

Local (Embedded Java)

Choose this model if any/all of the following are true

- ✅ One application and it is Java
- ✅ No network overhead (rather local JVM calls)

Avoid this model if any/all of the following are true

- ❌ Non-Java language in application
- ❌ Support for multiple applications
- ❌ Clustered environment (one application over multiple servers)

Required components

AppSensor Core
Represents central domain and interface specifications

Analysis Engine
Decides when an event becomes an attack and how to respond

Storage
Stores the data for the system (in-memory, file based, database)

Configuration
Represents your client (connect to server) and server (client info, detection points) configurations

Optional components

Access Controller (Required with REST or SOAP)
Ensures the client that is making requests has permission to do so

Reporting
Provides information about the system

How to start

Identify critical business functionality

Set detection points and thresholds

Determine proportional responses

Logging 'attacks'
is a good start

Implement

Duke Encounters

Welcome to **Duke Encounters**, the leading online platform for Java Duke spotting. This sample web application is using OWASP AppSensor for Application Intrusion Detection. The application is developed by Dominik Schadow, all source code is available on GitHub. This application is using an in-memory database to temporarily persist all information. Keep in mind that all entered information will be lost when restarting.

Latest Encounters

View all encounters or search for the ones you are interested in.

Event	Location	Date	Likelihood	Confirm
JavaOne 2014	San Francisco (USA)	2014-09-30	confirmed	
JavaOne 2013	San Francisco (USA)	2013-09-22	plausible	
JavaOne 2012	San Francisco (USA)	2012-10-01	plausible	
JavaOne 2011	San Francisco (USA)	2011-10-01	not confirmed	
JavaOne 2010	San Francisco (USA)	2010-09-22	not confirmed	

Demo

Only disabling accounts is useless when an attacker can create new ones

```
private void fireEvent() {
 if (user.isRegisteredToday()) {
  DetectionPoint dp = new DetectionPoint(
   INPUT_VALIDATION, "IE1-001");
  ids.addEvent(new Event(user, dp, detectionSystem));
 } else {
  DetectionPoint dp = new DetectionPoint(
   INPUT_VALIDATION, "IE1-002");
  ids.addEvent(new Event(user, dp, detectionSystem));
 }
}
```

```
private void fireEvent() {
 if (user.isRegisteredToday()) {
  DetectionPoint dp = new DetectionPoint(
   INPUT_VALIDATION, "IE1-001");
  ids.addEvent(new Event(user, dp, detectionSystem));
 } else {
  DetectionPoint dp = new DetectionPoint(
   INPUT_VALIDATION, "IE1-002");
  ids.addEvent(new Event(user, dp, detectionSystem));
 }
}
```

```
private void fireEvent() {
 if (user.isRegisteredToday()) {
  DetectionPoint dp = new DetectionPoint(
   INPUT_VALIDATION, "IE1-001");
  ids.addEvent(new Event(user, dp, detectionSystem));
 } else {
  DetectionPoint dp = new DetectionPoint(
   INPUT_VALIDATION, "IE1-002");
  ids.addEvent(new Event(user, dp, detectionSystem));
 }
}
```

Don’t lock admin account without backup unlock functionality

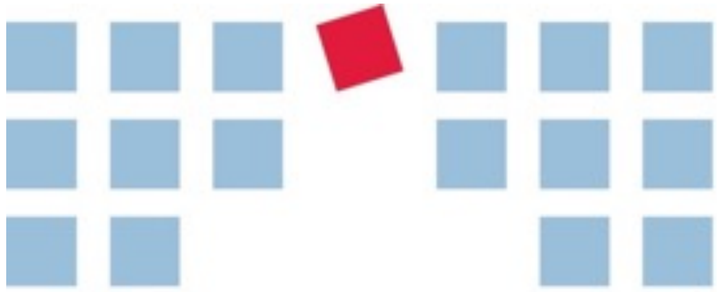
Summary

Immediately respond to ongoing attacks

Reduce impact of successful attacks

Learn about attacks and attackers

Develop securely and strike back



bridging IT

Koenigstr. 42
70173 Stuttgart
Germany

dominik.schadow@bridging-it.de
www.bridging-it.de

Blog blog.dominikschadow.de
Twitter @dschadow

Demo Projects
github.com/dschadow/ApplicationIntrusionDetection

OWASP AppSensor
appsensor.org

Detection Points
www.owasp.org/index.php/AppSensor_DetectionPoints

Pictures
www.dreamstime.com