[1/3] Integration of software in the “physical world”

- CPS – Cyber Physical Systems
- IoT – Internet-of-Things
- Computational capacity is omnipresent

Example:

- TRANSITION, From ad-hoc code development to code reuse through middleware for networked embedded control systems
Intelligence, relevant data all over the place…
- Context-aware computing unfolding beyond location and profile…
- Strong dependencies between sensing equipment, data processing entities and storage platforms
- Computational capacity is omnipresent, so is analytics…

Example:
- CAPRADS, A Context-Aware Platform for RApID Decision Support
Cloud Computing (*the trivial one*)

- Ultimately determining the delivery model of software and services…
- Flexible “software-defined” architectures to deal with rapid change, upgrading, reconfiguration, scaling etc.
- (But how about the attack surfaces?)

**Examples**

- **DeCoMAdS**: Deployment and Configuration Middleware for Adaptive Software-as-a-Service.
- **(DMS)$^2$**: Decentralized Data Management and Migration for Software-as-a-Service
Application Development today

- Despite all technology trends.....

AGILITY
Many technologies are available
Some still being developed, but on the horizon
Yet other are subject to a strategic investment (Still R&D)
Available?

- Dynamic Application Security Testing (DAST)
- … Static Application Security Testing (SAST)

- SIEM (Security Incident and event management)
- … Context-aware security (e.g. credentials that are requested/presented can depend on location).

- etc.
Note: Both development support and run time services/facilities to be integrated...
Following soon

- Mobile application security testing
- Web application firewalls
- Professional Services?
- Application Security as-a-service?
Many sources confirm ...

E.g. market analysts such as Gartner, Forrester etc.
Some of the heavy lifting

- DevOps & Security
- Protected Mobile Browsers
- (Runtime) Application Self-Protection

... and even further...
Illustration 1: *Isolating and Restricting Client-Side JavaScript* (towards a secure browser)

Featuring the PhD thesis of Dr. Steven Van Acker

January 6, 2015
Where to fix the problem?

1. JavaScript subsets and rewriting
2. Modifying the browser
3. Working with existing tools

Visitor

PuppyShelter.com

EvilSkeletor.com
Where to fix the problem?

1. JavaScript subsets and rewriting

PuppyShelter.com

Visitor

EvilSkeletor.com
JavaScript subsets and rewriting

- Main idea: analyze JavaScript before executing it, rewriting if necessary
- Examples: Caja, FBJS, ADsafe, BrowserShield, …
- Unfortunately:
  - Analyzing JavaScript is difficult. Using a JavaScript subset makes it easier but requires effort from third-party
  - Rewriting JavaScript changes architecture of the Web
Where to fix the problem?

2. Modifying the browser

PuppyShelter.com

Visitor

EvilSkeletor.com
Modifying the browser

Visitor

BEEP, ConScript, WebJail, AdSentry, …
WebJail: Least-privilege Integration of Third-party Components in Web Mashups

Steven Van Acker, Philippe De Ryck, Lieven Desmet, Frank Piessens, Wouter Joosen
ACSAC 2011
WebJail: main idea

- Restrict sensitive JavaScript functionality in the DOM of an iframe
- An advice function intercepts calls to a DOM function and mediates access
- All access-paths go through the advice function
- Enforced in the browser, advice is locked away safely
WebJail: policies

- Easy to use policy language
- All JavaScript functionality divided into 9 categories:
  - DOM Access
  - Cookies
  - External communication
  - Inter-frame communication
  - Client-side storage
  - UI and rendering
  - Media
  - Geolocation
  - Device access

```json
{
  "framecomm": "yes",
  "extcomm": [ "google.com", "youtube.com" ],
  "device": "no"
}
```
WebJail: architecture

Diagram showing the architecture layers:
- Policy layer
- Advice construction layer
- Deep aspect weaving layer

Symbols:
- Policy file
- Category
- API
- API Function
- Advice function
- Advised function
WebJail: conclusion

- WebJail is a viable JavaScript sandbox
  - Full mediation
  - Fast
- Unfortunately:
  - Deploying a browser modification to all browsers on the Web is hard
  - “Just get the modification adopted by W3C so all browsers implement it” → not so easy…
Where to fix the problem?

3. Working with existing tools

PuppyShelter.com
Visitor
EvilSkeletor.com

SecAppDev 2015
JSand: Complete client-side sandboxing of third-party JavaScript without browser modifications.

Pieter Agten, Yoran Brondsema, Steven Van Acker, Phu Phung, Lieven Desmet, Frank Piessens

ACSAC 2012
JSand: object-capability env.

- **Object capability environment:**
  - All functionality is encapsulated in objects
  - References to those objects can not be forged
  - Without reference to a certain object, there is no access to its functionality

- **E.g. window.alert()**
  - alert is a property of the window object
  - Without access to the window object, alert() can not be used

- **Secure ECMAScript is object-capability safe**
  - Subset of JavaScript strict mode
JSand: Under the hood

- Download third-party script directly to browser
- Load script in **isolated** object-capability environment using Google’s Secure ECMAScript
- Enable access to outside using *membrane* around DOM
  - Policy determines permitted operations
JSand: Conclusion

- JSand is also a viable sandboxing solution
  - Full mediation
  - Works out-of-the-box on modern browsers
- Unfortunately:
  - Reusing functionality that was not intended for sandboxing results in unwanted performance hit
Observations

- There is no silver bullet (yet)
- Reusing currently standardized functionality is not optimal
  - E.g. performance overhead
- Specialized JavaScript sandboxing functionality is required
  - Proof of concept as browser modification
  - But in long run, functionality must be standardized
Illustration 2: Security Primitives for Protected Module Architectures

Featuring the PhD thesis of Dr. Raoul Strackx
December 17, 2014
Protected Module Architectures:

- Low-level security architectures that implement an “inverse sandbox”: protect a module from a buggy or malicious environment
  - E.g. run code securely even on top of a kernel infected with malware
Emerging technology: PMA’s

- Implementations
  - Pioneering work by Parno et. al. at CMU: the Flicker system
    - [https://sparrow.ece.cmu.edu/group/flicker.html](https://sparrow.ece.cmu.edu/group/flicker.html)
    - Bryan Parno was awarded the ACM 2010 doctoral thesis award for this work
  - Follow-up implementations, including several from iMinds:
    - Fides (Strackx et al, CCS 2012), Sancus (Noorman et al., Usenix Sec 2013)
  - INTEL publicly announced their implementation quite a while ago (snclaves in SGX)
Protected module architecture (simplified)

- Modules consist of:
  - A code section, with designated entry points
  - A data section (also containing control data)

- The PMA:
  - Controls creation/deletion of modules
  - Enforces a PC-based access control model
Some Achievements

- How can Protected Module Architectures efficiently, securely and reliably persist state?
- What is the minimal hardware support required to implement PMA’s:
  - That support remote attestation
  - That support state continuity
  - That do not need software in the TCB
Research challenges ahead

- How do we offer higher-level abstractions for these low-level security architectures?
  - Key idea: maintain the modularity properties of source code at machine code level by secure compilation.

- How do we provide assurance of the correctness of the protected module itself?
  - These modules might be small enough to be amenable to formal verification.
This type of work may lift self-protection to the next level
Illustration 3: Amusa
Access control middleware for multi-tenant SaaS applications
Goal

- Combine policies securely
- Enforce at run-time
Three-layered access control mgmt

<table>
<thead>
<tr>
<th>Tenants</th>
<th>Large Bank</th>
<th>Press Agency</th>
</tr>
</thead>
<tbody>
<tr>
<td>subj.assigned_customers</td>
<td>subj.region</td>
<td></td>
</tr>
<tr>
<td>eDocs</td>
<td>subj.email, subj.tenant_credit, res.sender</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Amusa</th>
<th>Large Bank</th>
<th>Press Agency</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Deny if not res.owner in subj.assigned_customers</td>
<td>Deny if subj.region != &quot;Europe&quot;</td>
</tr>
<tr>
<td></td>
<td>Override isolation if subj.tenant == &quot;PartnerA&quot;</td>
<td></td>
</tr>
<tr>
<td>eDocs</td>
<td>Deny if subj.tenant_credit &lt; action.cost</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Override isolation if res.owner in subj.reseller_tenants</td>
<td></td>
</tr>
<tr>
<td>Amusa</td>
<td>Default tenant isolation policy</td>
<td></td>
</tr>
</tbody>
</table>
Logical architecture

Legend:
- User interface
- Authorization flow
- Configuration flow

Application

Application end-user

Tenant admin

Provider admin

Policy Decision Point

Application resources

Attribute definitions

Subject attributes

AuthN Endpoint

Admin Dashboard

Admin Dashboard

Amusa
STAPL
The Simple Tree-structure
Attribute-based Policy Language
A note on the relative ease of specifying policies

<table>
<thead>
<tr>
<th></th>
<th>Attr. def.</th>
<th>Obl. def.</th>
<th>Pol. spec.</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>E-health</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>XACML</td>
<td>-</td>
<td>-</td>
<td>706</td>
<td>706 (100%)</td>
</tr>
<tr>
<td>ALFA</td>
<td>168</td>
<td>3</td>
<td>259</td>
<td>430 (60.9%)</td>
</tr>
<tr>
<td>STAPL</td>
<td>27</td>
<td>4</td>
<td>84</td>
<td>115 (16.3%)</td>
</tr>
<tr>
<td><strong>E-docs</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>XACML</td>
<td>-</td>
<td>-</td>
<td>1332</td>
<td>1332 (100%)</td>
</tr>
<tr>
<td>ALFA</td>
<td>175</td>
<td>3</td>
<td>514</td>
<td>692 (52.0%)</td>
</tr>
<tr>
<td>STAPL</td>
<td>31</td>
<td>4</td>
<td>196</td>
<td>231 (17.3%)</td>
</tr>
</tbody>
</table>
Focus on multi-tenant IAM

Main technology:
- policy-based access control with attribute-based tree-structured policies
- STAPL: policy language (DSL)
  - suited for extending with new technologies

.. Clearly WIP.
Attitude of the market

- Security Provider side: point solutions and network level technology taking a lot of spotlight.

- Software Vendor (ISV side): managing performance indicators (e.g. #bugs found) may not truly support application security

- Agility remains obviously crucial….

So is there any room for Secure SDLC? 😊ristol
In this corner, we have firewalls, encryption, antivirus software, etc.

And in this corner, we have Dave!!
MUST consider:

4 angles
(1) Life Cycle Support

(not XP)
(2) Expressive Power

Remember policy languages…
(3) Composition & Transformation

Automation is crucial for cost purposes and robustness…
(4) Dev Ops...

Deployment (configurations etc.) must/will become an integrated part of software
Illustration 1: RE
Requirements Engineering
Privacy threats in software architectures
Development lifecycle

- white
- some storage space
- sleek design

Requirements ➔ Architecture ➔ Implementation

- white
- sleek design

KITCHEN SOFTWARE

23/02/2015

SecAppDev 2015
LINDDUN Threat modeling

- Eliciting threats
  - Related to **Linkability**, **Identifiability**, **Non-repudiation**, **Detectability**, **Disclosure of information**, **Unawareness**, **Non-compliance**

- Model of the system highlighting the assets
  - Components (processing, data) and info flows

- Finding flaws that could lead to attacks

- “Not unlike” Microsoft’s **STRIDE**
MUC 05: Summary

A researcher or other insider with malicious intent links PHR data (or user data)

Primary mis-actor: unskilled insider (authenticated user, e.g. researcher)

Basic path:

bf1. The misactor performs a set of targeted queries on the PHR data or user data store and retrieves very detailed results

bf2. The misactor links the results of the queries together (e.g. based on medication which is usually combined, medical conditions which occur together, or pseudo-identifiers like street and age)

Consequence: By combining the query results, the misactor has access to more information about the patient than anticipated

Reference to threat tree node(s): L_ds2, L_e2

Parent threat tree(s): L_ds, I_ds

DFD element(s): 5.1 PHR data, 5.2 user data

Remarks:
r1. This threat can be used as precondition for the identifiability threat at the data store (T03 - Identifying a patient from his PHR data)
LINDDUN in the wild

In privacy talks

Applied in European projects

Use-cases definition and threat analysis

Deliverable 5.3

Project Title: Building data bridges between biological and medical infrastructures in Europe
Project Acronym: BioMedBridges
Grant agreement no.: 284209
Research infrastructures, FP7 Capacities Specific Programme: [INFRA-2011-2.3.2.] Implementation of common solutions for a cluster of ESFRI infrastructures in the field of "Life sciences"
Deliverable title: Report describing the security architecture and framework
Actual delivery date: 30 June 2014

Contents

4. Security and privacy concepts and definitions ........................................... 14
4.1. Threat, vulnerability, and risk ............................................................ 14
4.2. Security threat modelling using STRIDE ........................................... 16
4.2.1. Threats and security properties addressed by STRIDE .................... 17
4.3. Privacy threat modelling using LINDDUN ........................................... 18
4.3.1. Threats and privacy properties addressed by LINDDUN ................. 19
Ad Interim - Summary of evaluation

- **Advantages**
  - Acceptable *correctness* rate
  - Relatively *easy to learn* and apply
  - LINDDUN threat tree *catalog* is useful

- Good *coverage* of privacy threats

- **Room for improvement**
Illustration 2: Source Code analysis – vulnerability prediction
impact of software quality on security

- Specialists: verification technology
  - Direct assessment (A)

- For any developer
  - Indirect assessment (B)
VeriFast

C or Java source code

Specification

Proof hints

Guarantees that program
- has no buffer overflows
- has no integer overflows
- has no data races
- uses APIs correctly
- satisfies specification

"0 errors found"

Symbolic execution trace showing error

User can step through trace and inspect symbolic states

~ 1s
VeriFast: verified programs - cases

- Fine-grained concurrent data structures
  - Functional correctness
- JavaCard applets (incl. for Belgian eID card)
  - Crash-freedom, safe API usage
- Linux device drivers
  Memory safety, data-race-freedom, safe API usage
- Embedded software (for Telefonica home gateway)
  - Memory safety, data-race-freedom, safe API usage
- Cryptographic protocol implementations (RPC, Needham-Schroeder-Lowe)
<B> Fault Prediction, based on Text Mining

Software Quality @ Development time
Research question

Can we build a (good quality) classifier that predicts vulnerable components in C++ applications?

Idea: Analyze the tokens in each component's code (e.g., if, while, variable names) and use these as predictors.
Build prediction model (using 1 version)

Test performance of prediction model (in each of the following 8 version)
Benchmark

- Find at least 80% of the components containing vulnerabilities (cost) by inspecting at most 20% of the application components (benefit)
Results

- We exceedingly meet the benchmark
  - For all the “future” versions

- Better than best results in the state-of-the-art (i.e., Shin et al., TSE 37(6), 2011)
And now... Reaching out!

Which problems are perceived to be of the highest priority? We start an anonymous survey of ISV’s in Flanders and beyond (Q2 2015)
Challenges Summarized

Full life cycle support must become agile, but it remains high priority. (Part 2)

This cannot be achieved without managing the concept of risk.

New techniques can and should contribute to reducing the overall cost.

This must be pursued while dealing with all other trends of these interesting times…. (Part 1)
Thank You!
Thank them!

Jasper Bogaerts, Maarten Decat, Ming Deng, Philippe De Ryck, Lieven Desmet, Thomas Heyman, Aram Hovsepyan, Bart Jacobs, Bert Lagaisse, Fabio Massacci, Sam Michiels, Jasper Moeys, Frank Piessens, Bart Preneel, Davy Preuveneers, Riccardo Scandariato, Steven Van Acker, Dimitri Van Landuyt, James Walden, Kim Wuyts, Koen Yskout, …