Prometheus: Extracting WebInject signatures from Information Stealers

Andrea Continella, Stefano Zanero, Federico Maggi

Politecnico di Milano, Italy

NCG2015 @ Microsoft Research

06/12/2015

NECST laboratory

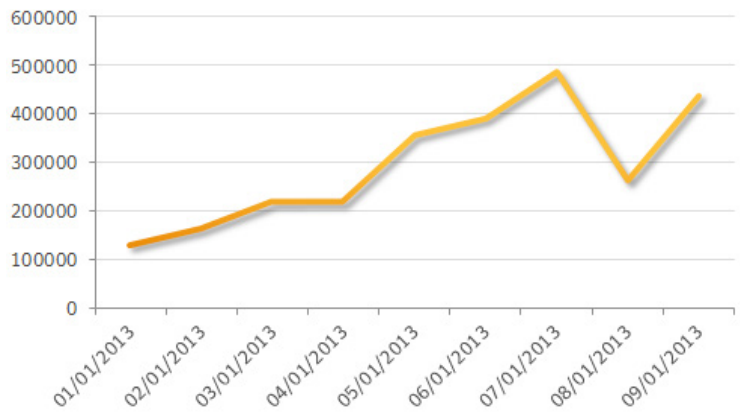
Introduction

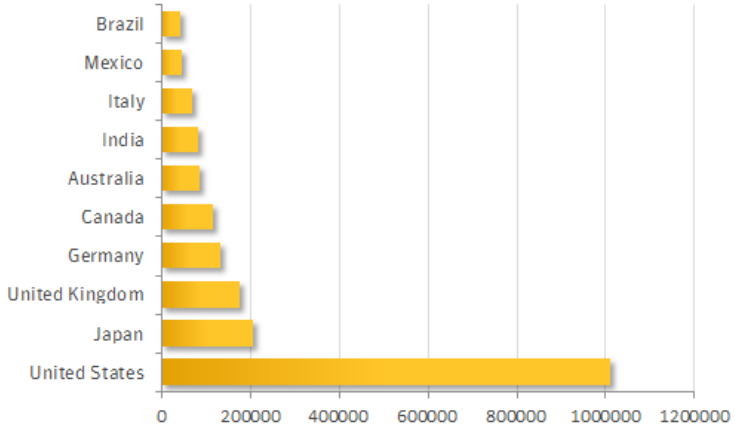
Information Stealers

- What they are
  - Malware that steals credentials such as usernames, passwords, and second factors of authentication
  - Also known as "banking trojans": steal banking credentials and perform online financial frauds
- ZeuS (2007), SpyEye (2011), Citadel (2012), are the most notorious
- What they do
  - Intercept credentials and private information submitted to web forms
  - Harvest and steal files
  - Hijack browser session
  - Use the victim's device as a proxy

Why do we care

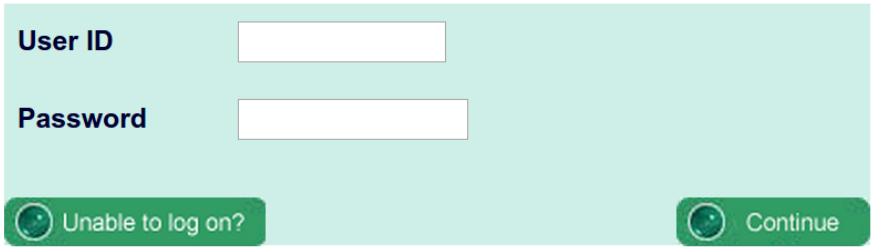
- They affect directly the victims' "pockets"
- Billions of dollars stolen through banking trojans
- New variants are constantly being introduced to evade modern defense mechanisms
- They can be easily purchased on underground markets
- Low detection rate: 40.11% (May 15, 2015, ZeuS Tracker)

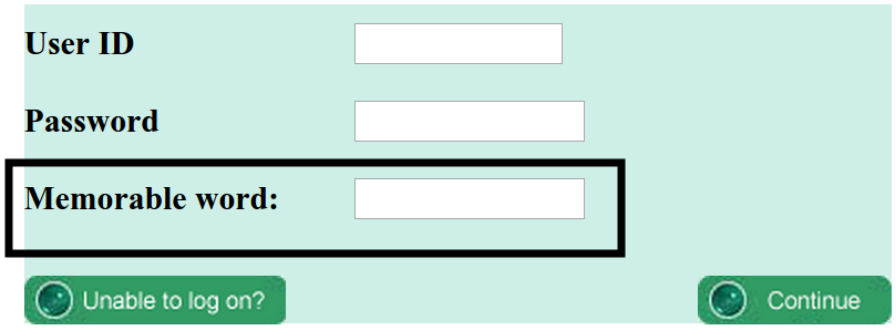




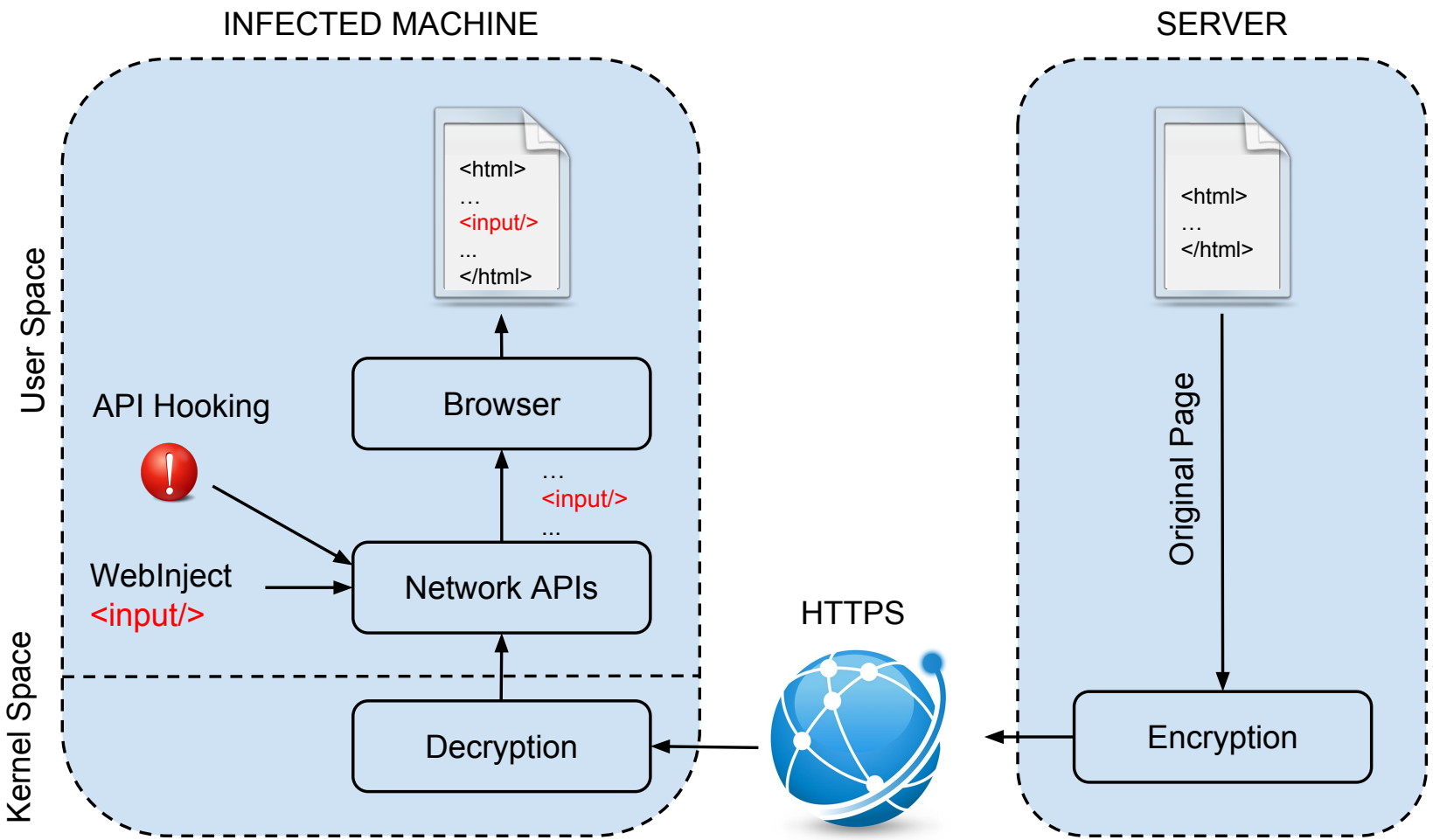
Man in the Browser and WebInject

- **API hooking:** intercept all the data going through the browser even when the connection is encrypted (Man in the Browser)
- **WebInject:** manipulate and modify web pages
- **Goal:** make the victim believe that the web page is legitimately asking for the second factor of authentication or any other private information





Hooking Mechanism



Example of ZeuS WebInject configuration file

```
set_url https://www.wellsfargo.com/*
data_before
    <span class="mozcloak"><input type="password" *</span>
data_end
data_inject
    <br><strong>
    <label for="atmpin">ATM PIN</label>:</strong><br />
    <span class="mozcloak"><input type="password" accesskey="A"
     id="atmpin" name="USpass" size="13" maxlength="14"
     style="width:147px" tabindex="2" /></span>
data_end
```

State of the art and Limitations

- Malware executables are packed and obfuscated
- Configuration files are encrypted
- Different families, versions, browsers, API-hooking methods, operating systems
- Slight changes require manual reverse engineering

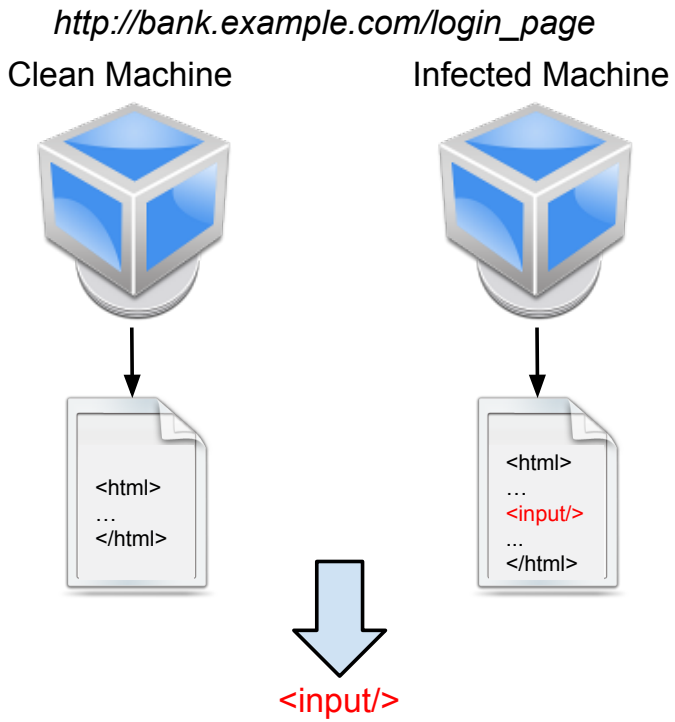
Goals

- Capture the inherent behavior of the targeted family
- Compile behavioral signatures, automatically
- Minimize reliance upon the implementation details
- No reverse engineering required

Prometheus

Proposed approach: Web-page differential analysis

- The key idea is to analyze banking trojans exploiting the visible DOM modifications that they cause inside the HTML pages

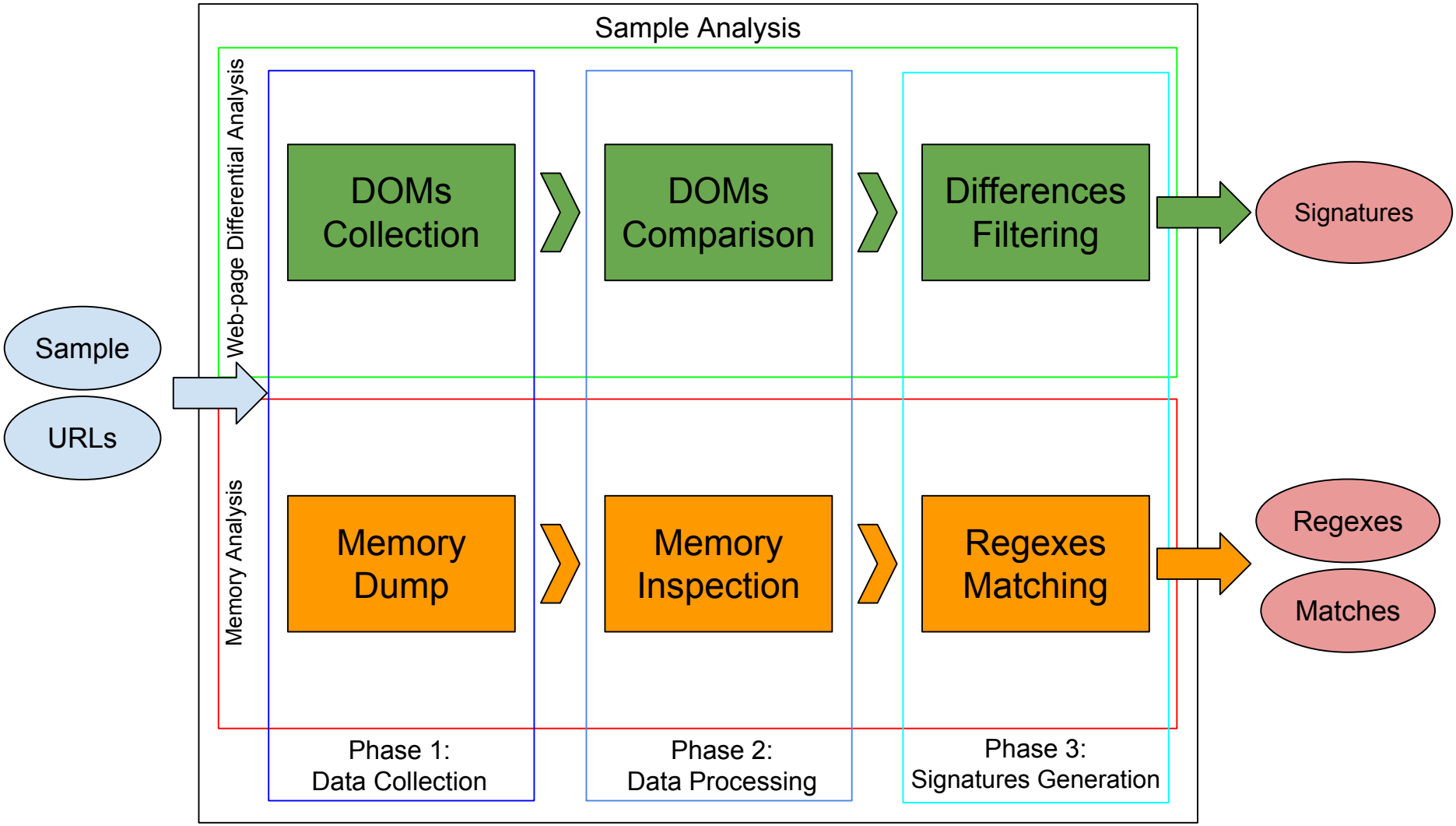


- **Challenge:** Not all differences are malicious! We need multiple machines and proper filters

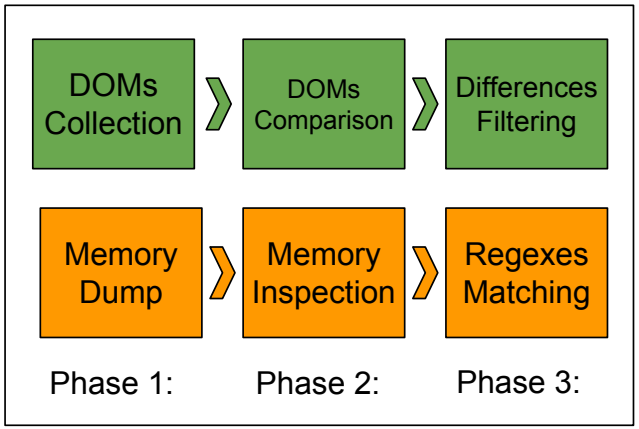
Proposed approach

- **Web-page Differential Analysis**
  - generate signatures identifying malicious injections
- **Memory Forensics**
  - extract the WebInject targets from infected memory dumps to validate signatures

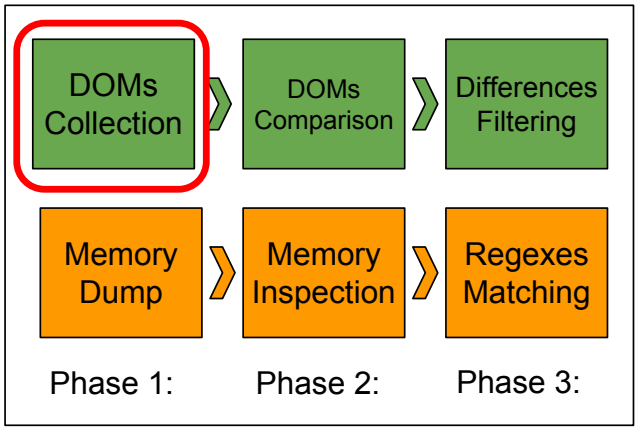
System Overview

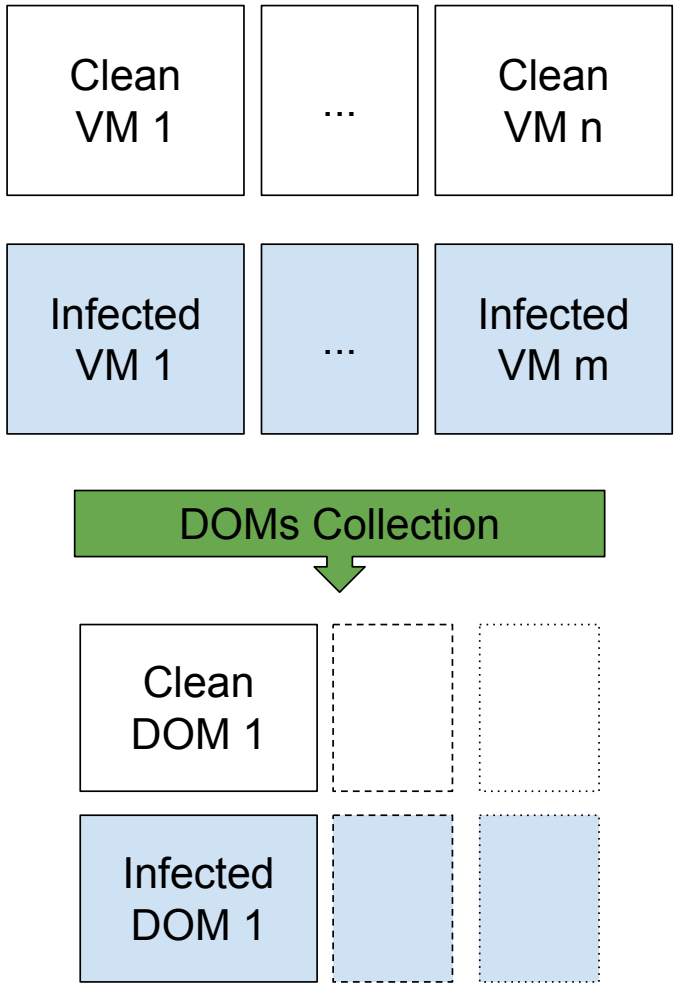


System Overview - DOMs Collection

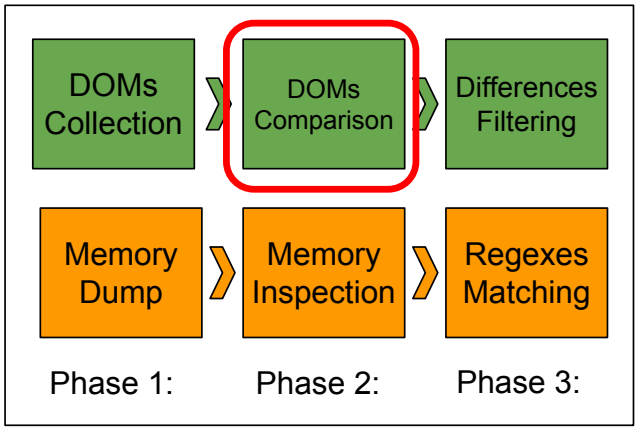


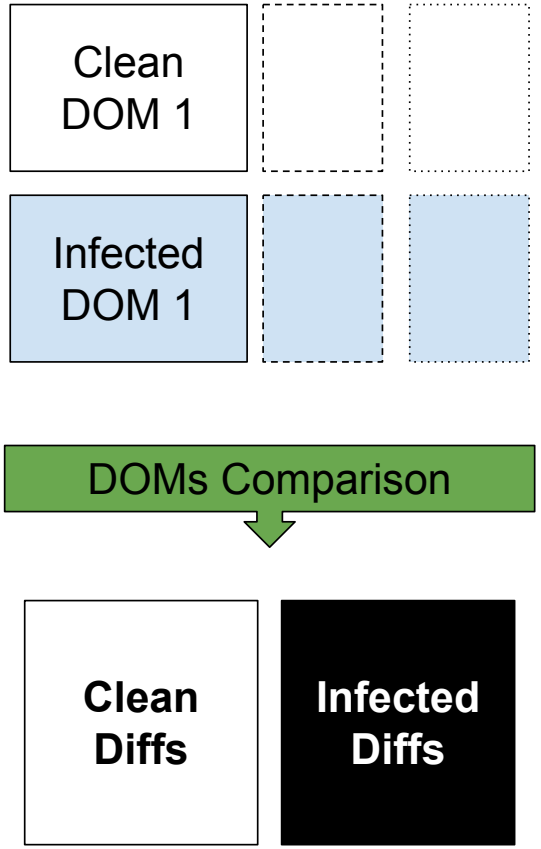
System Overview - DOMs Collection



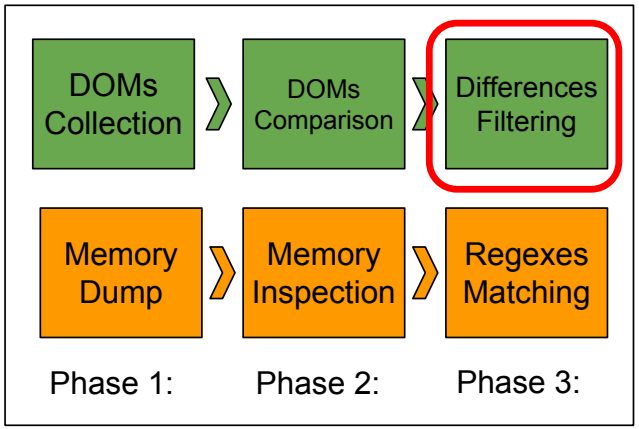


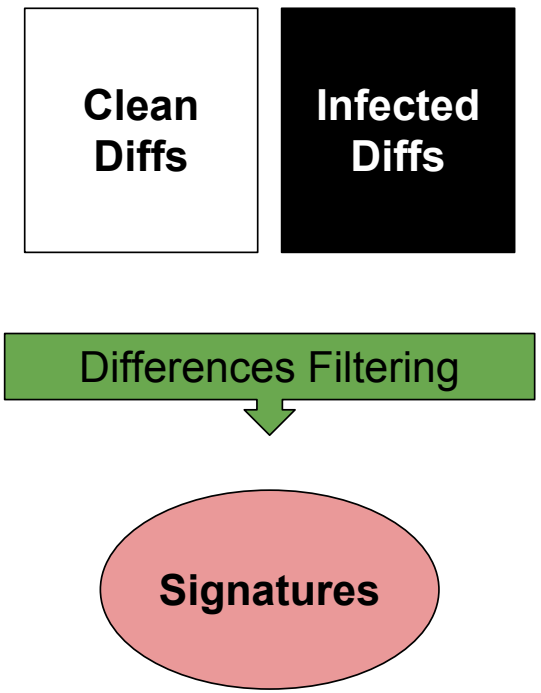
System Overview - DOMs Comparison



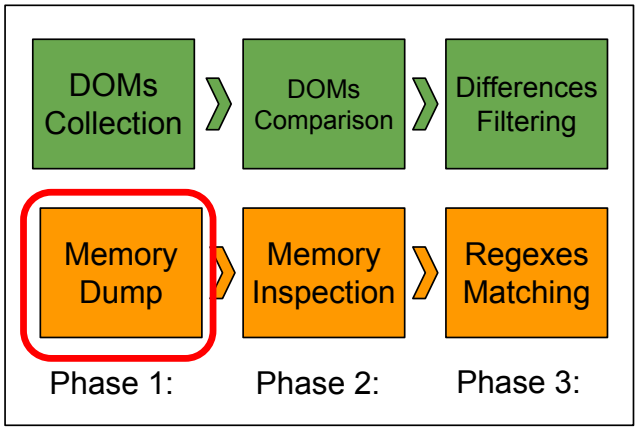


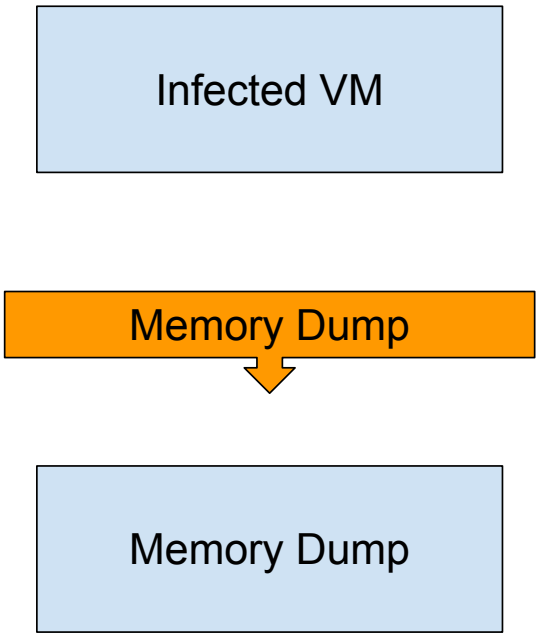
System Overview - Differences Filtering



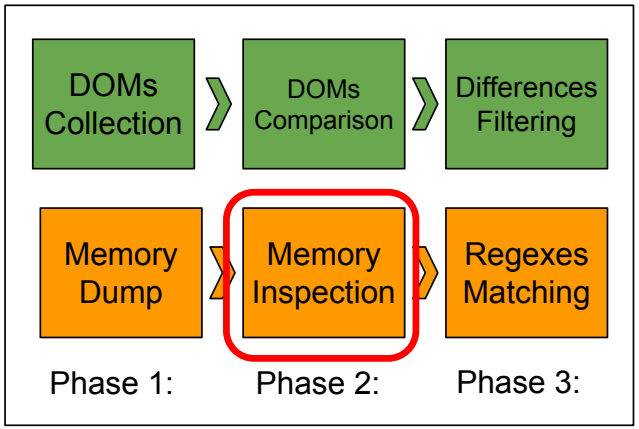


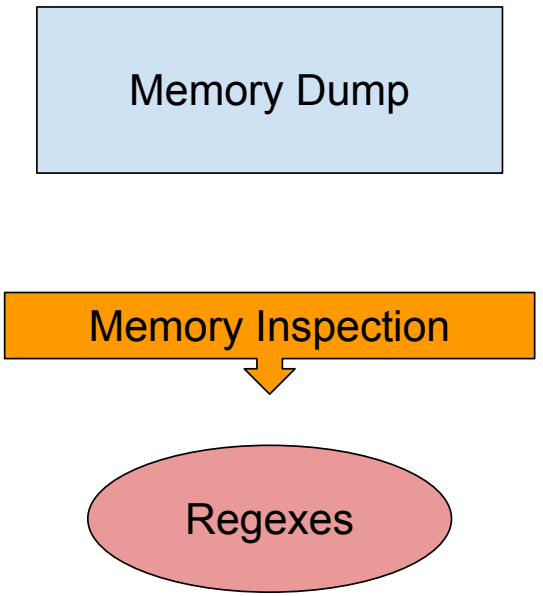
System Overview - Memory Dump



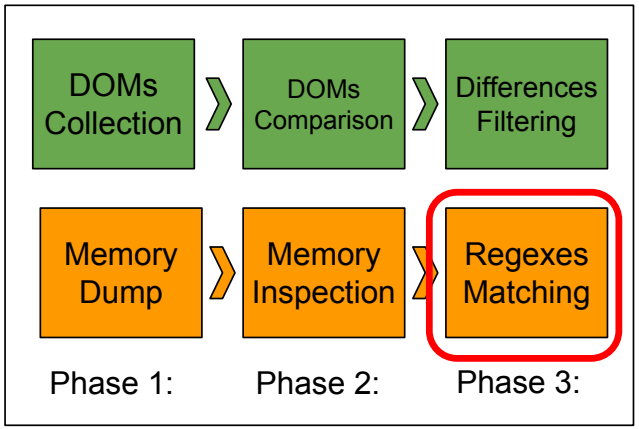


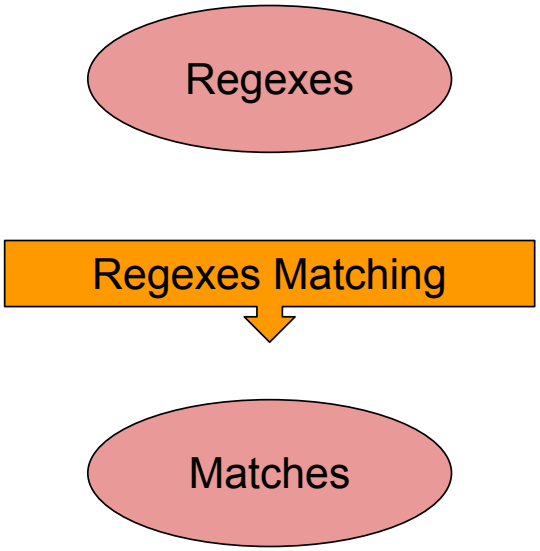
System Overview - Memory Inspection





System Overview - Regexes Matching





System Overview - Regexes Matching

```
set_url https://www.wellsfargo.com/*
data_before
   <span class=‘‘mozcloak’’><input type=‘‘password’’ *</span>
data_end
data_inject
   <br><strong>
   <label for=‘‘atmpin’’>ATM PIN</label>:</strong><br />
   <span class=‘‘mozcloak’’><input type=‘‘password’’
accesskey=‘‘A’’
    id=‘‘atmpin’’ name=‘‘USpass’’ size=‘‘13’’ maxlength=‘‘14’’
    style=‘‘width:147px’’ tabindex=‘‘2’’ /></span>
data_end
```

Benign vs. Malicious differences

Each difference is composed by three elements

- **Type:** The nature of the difference (node insertion, node modification..)
- **XPath:** The XML path to the node that is affected by the difference (e.g., `/html/body/form[1]/input[3]`)
- **Content:** The value of the difference (e.g., in the case of a node insertion the content is the HTML node inserted with all its attributes)

Multi-VM diffing: Ignore diffs appearing in clean machines.

Diffing Heuristics

Heuristic 1: Ignore harmless differences

- pure node or attribute **reordering**,

```
<input type=“pass” class=“foo” />  <input class=“foo” type=“pass” />
```

- **styling** changes (e.g., value, width, height, sizset, alt)

Diffing Heuristics

Heuristic 2: Filter benign nodes with frequent changes

- Filter out differences generated by clean vs. clean machine

Clean VM 1:

```
...
<body>
  ...
  <a href=page?session=1111>
  ...
</body>
...
```

Clean VM 2:

```
...
<body>
  ...
  <a href=page?session=2222>
  ...
</body>
...
```

Diffing Heuristics

Heuristic 3: Malicious injection

- Filter out the infected differences that are not present in the majority of the infected DOMs (more than ε)

Signature Example

```
{
   “signatures”:  [
      {
         “type”:  3,
         “xpath”: “/html[1]/body[1]/table[3]/tr[1]/form[1]/input[13]”,
         “content”: “<input name=OTP type=password/>”
      }
   ]
}
```

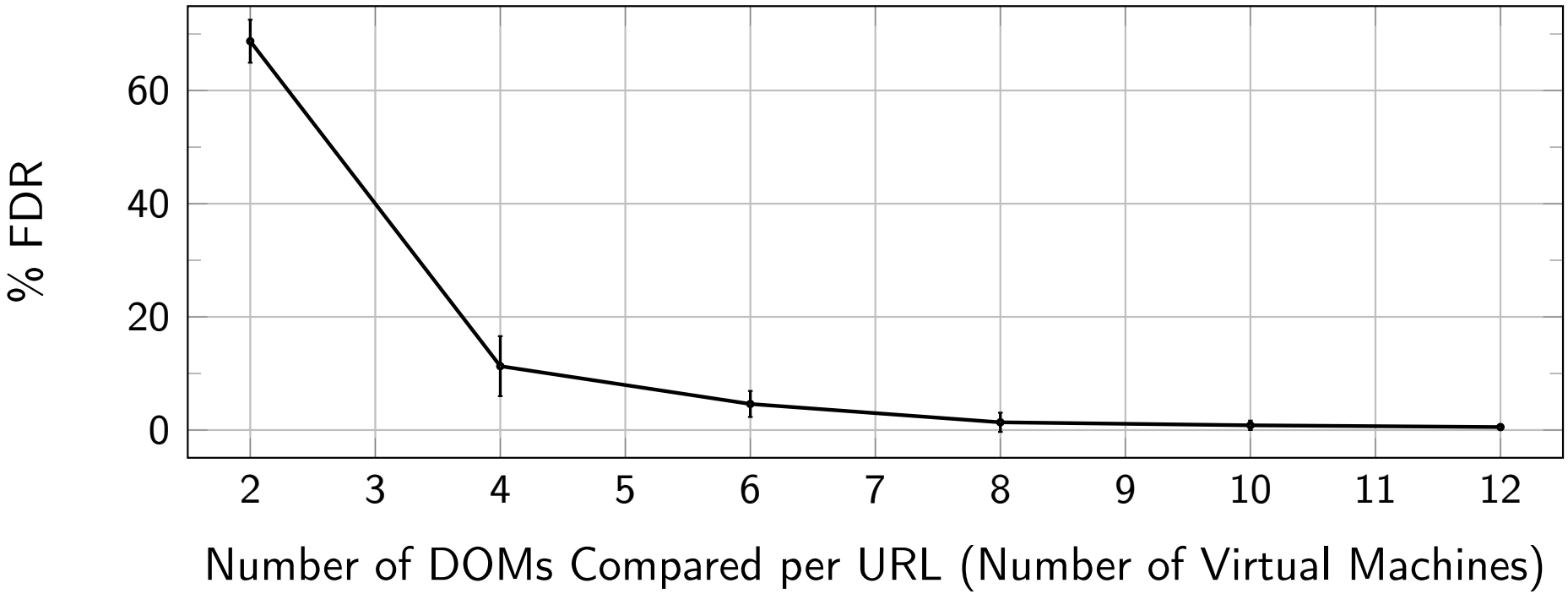
Evaluation

Datasets

- We evaluated Prometheus on a dataset of 196 distinct samples of ZeuS
  - However only 135 were active
- We analyzed 68 URLs of banking websites extracted from real configuration files

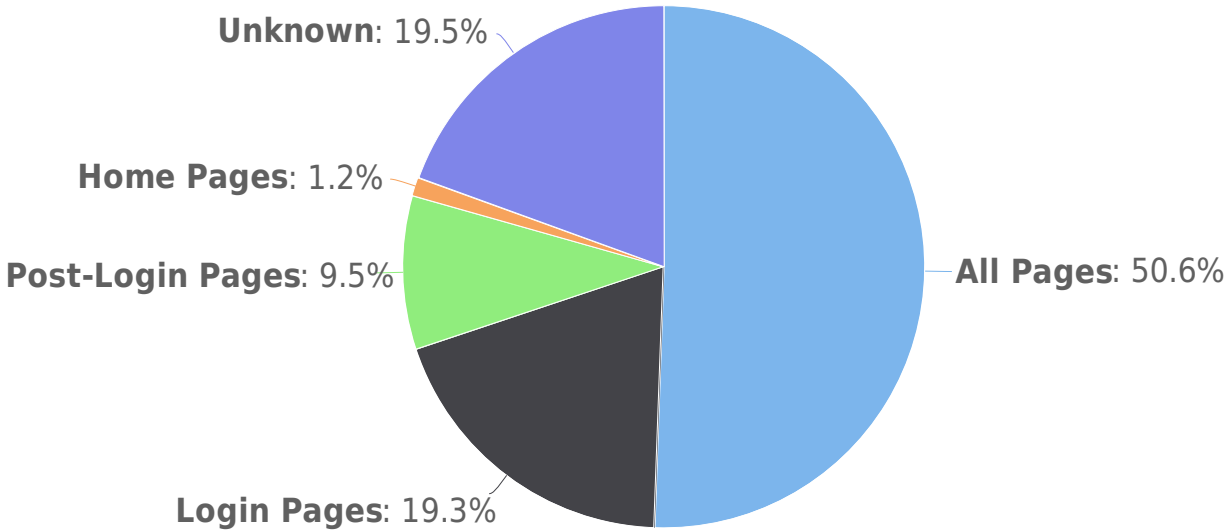
False differences

- The overall false difference rate is 0.70%
- Validation through memory forensic reduces the false difference rate to 0.26%



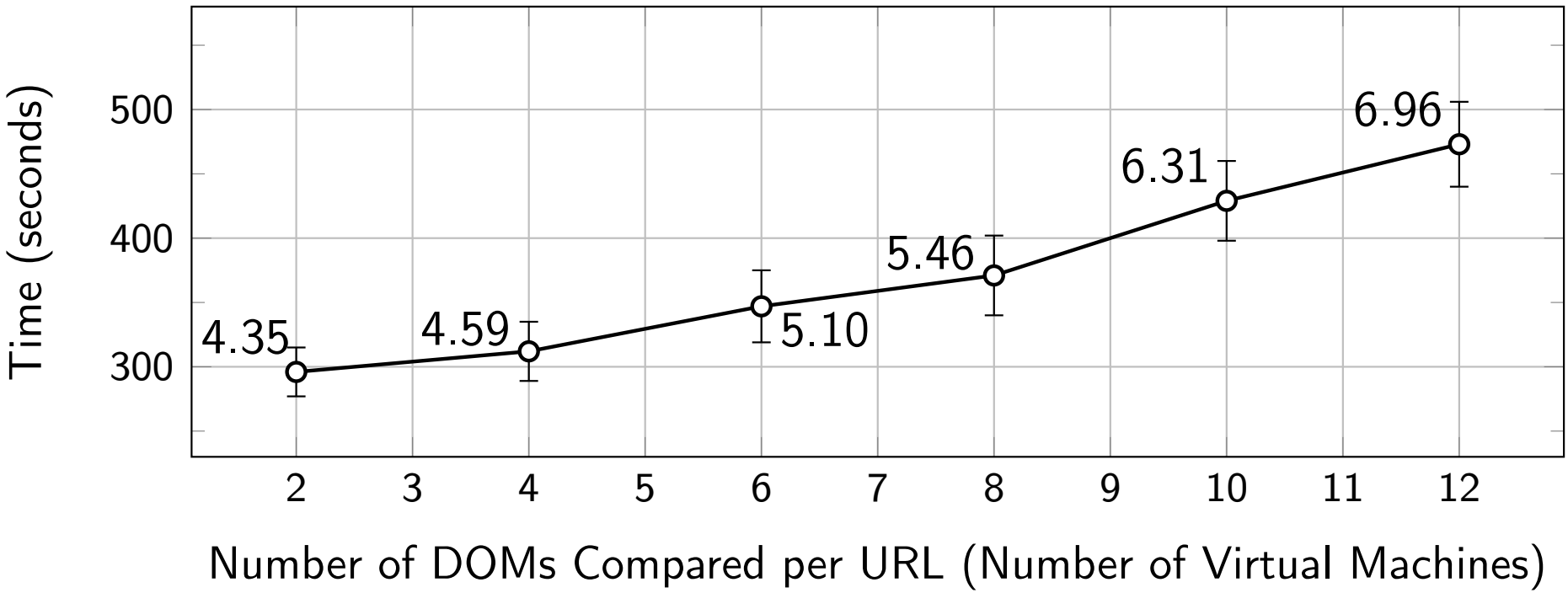
Memory analysis

- We successfully extracted 694 distinct regular expressions through memory forensics
- True injections were present in URLs that matched some of the extracted regular expressions
- We classified regular expressions according to the kind of targeted pages (ex. all pages: `*domain.com/*`)



Performance

- We deployed Prometheus on a 2.0GHz, 8-core Intel machine with 24GB of RAM



Conclusions

Limitations

- Injections that overlap perfectly with existing nodes are discarded
  - hard to produce without disrupting the page
- Samples that perform dynamically changing content injections can evade our system
  - did not see any, so far

Future Works

- Design of client and server side protections together with a proper signature matching algorithm
- Focus on more advanced uses of WebInjects
  - WebInjects performing dynamic-content injections
  - WebInjects performing attacks that may not result in DOM modifications

Conclusions

- Simple yet very **effective** technique
- **No** reverse engineering required
- **Independent** from implementation, language, platform

THANK YOU!

andrea.continella@polimi.it - @_conand

QUESTIONS?