

# **KIZZLE**

## **A SIGNATURE COMPILER FOR DETECTING EXPLOIT KITS**

**Ben Stock (CISPA, Saarland University)**

Ben Livshits (Microsoft Research)

Ben Zorn (Microsoft Research)

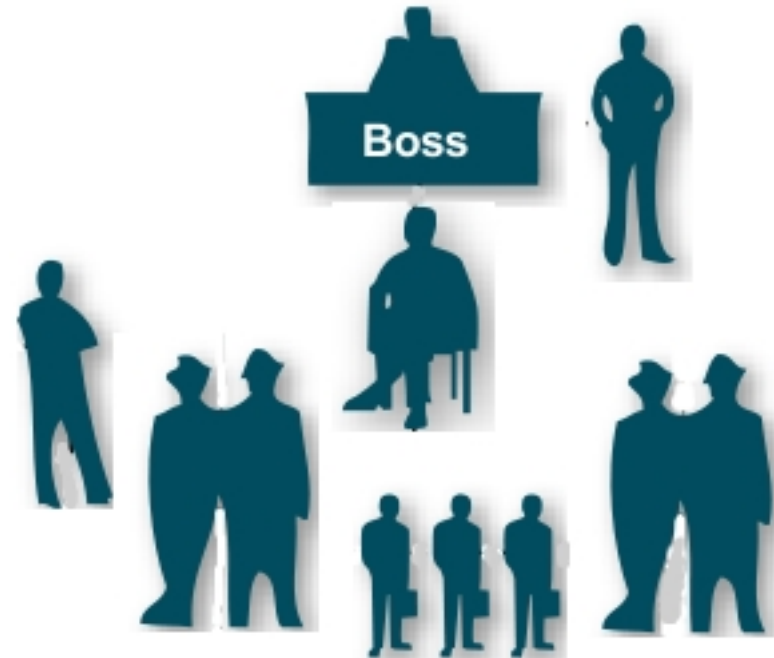
`/Ben [chiklnorstvz]{4,8}/i`

# EXPLOIT KITS: CONSOLIDATING MALWARE PRODUCTION

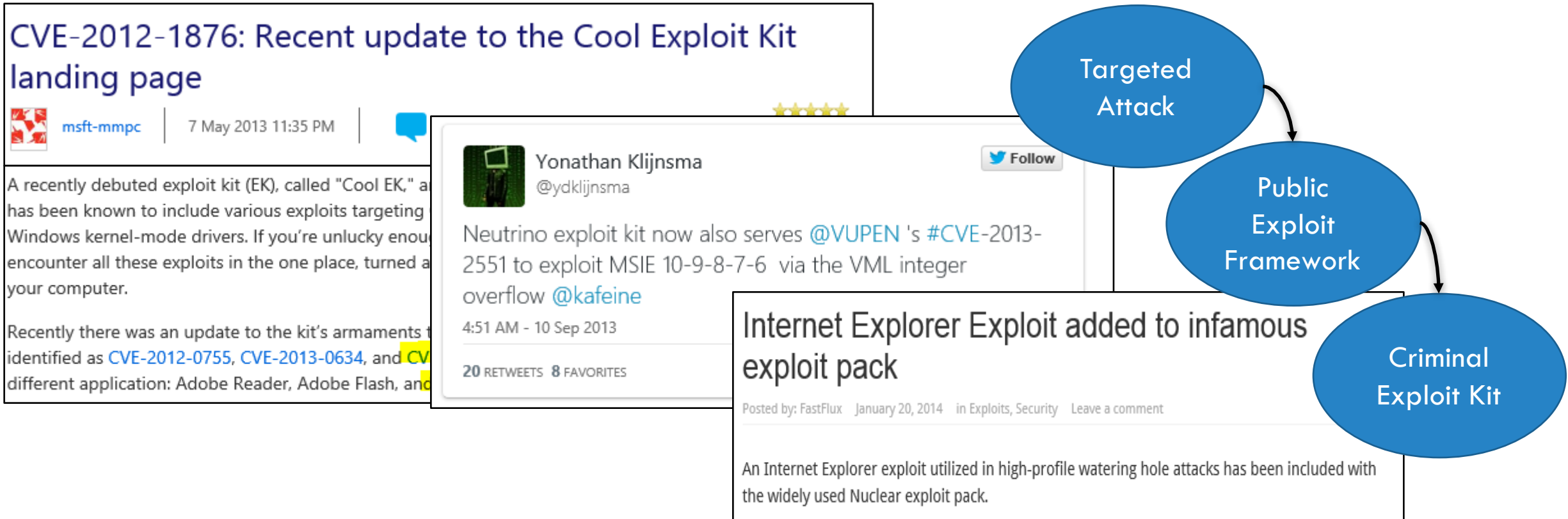
2006



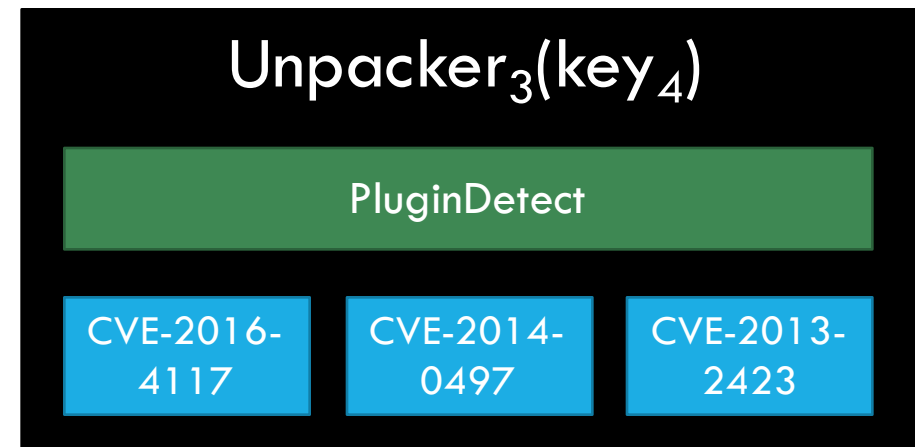
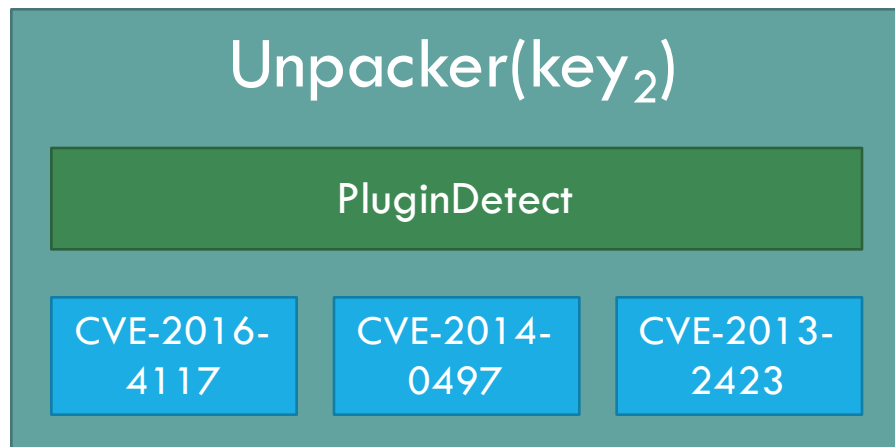
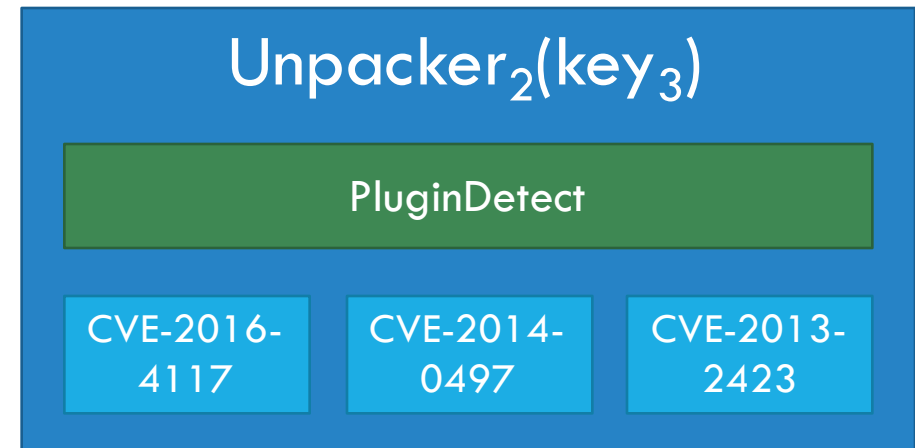
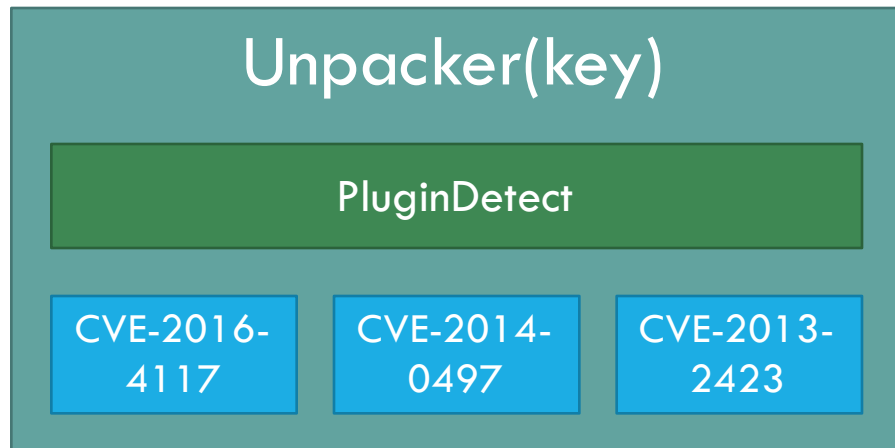
2016



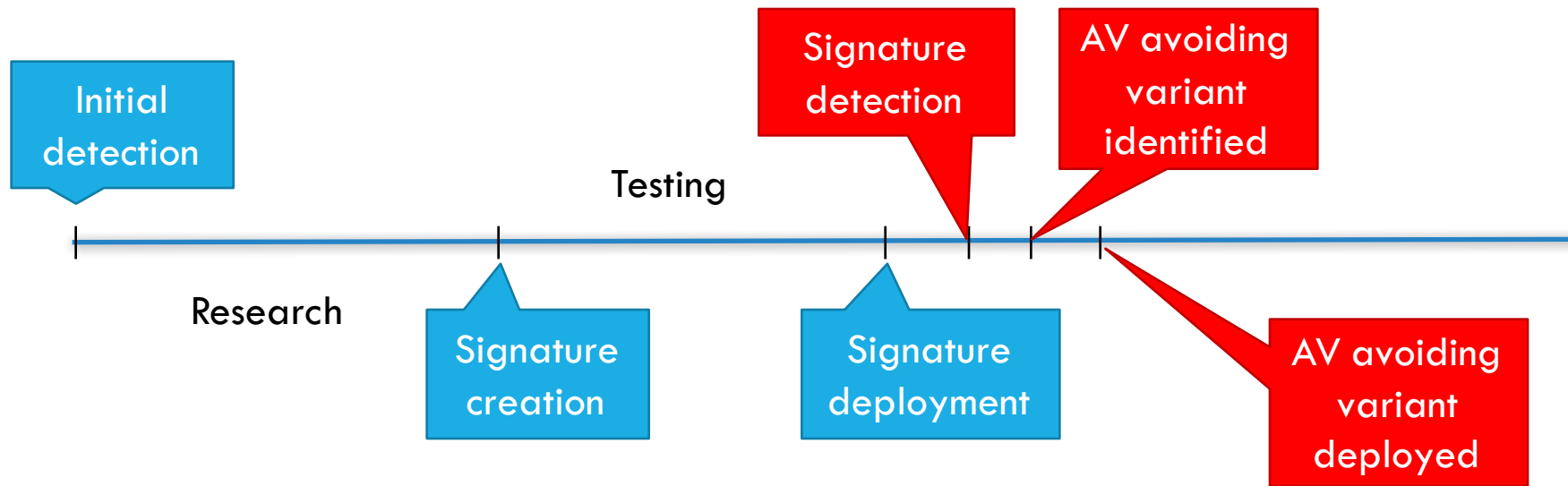
# EXPLOIT KITS (EKS) PROVIDE AN “EXPLOIT-AS-A-SERVICE”



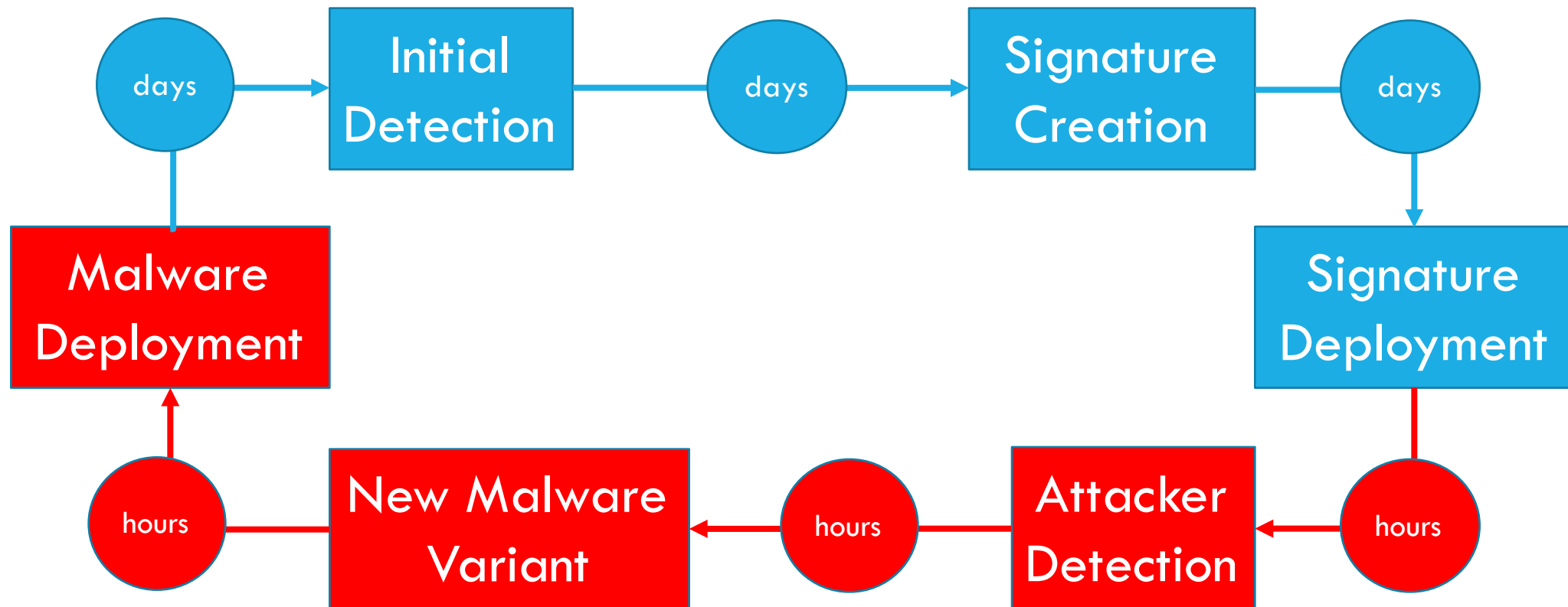
# STRUCTURE OF A TYPICAL EXPLOIT KIT



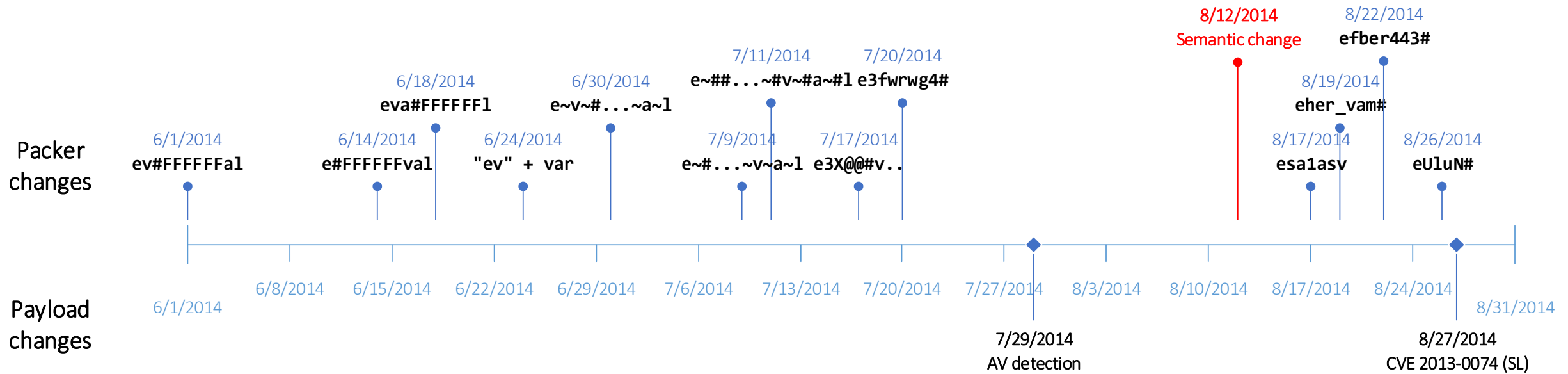
# MALWARE ECOSYSTEM AND AV SIGNATURES



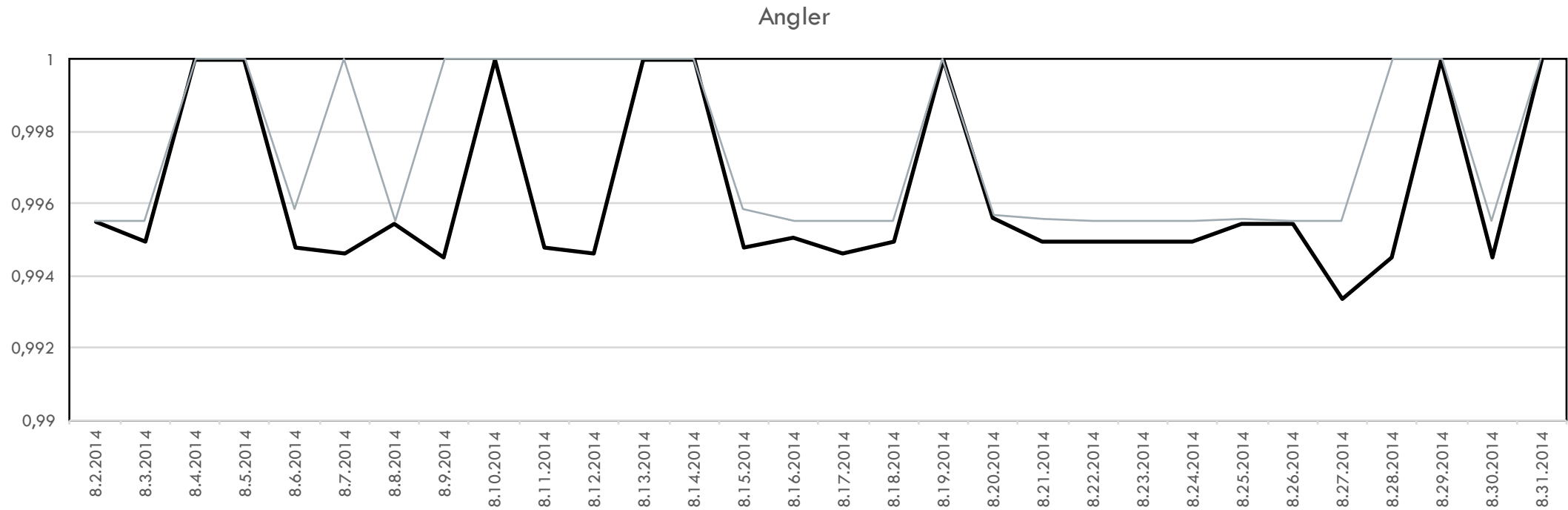
# ADVERSERIAL CYCLE FOR EXPLOIT KITS



# EXPLOIT KIT EVOLUTION: NUCLEAR



# A GREAT DEAL OF CODE REUSE





# EXPLOIT KITS: KEY INSIGHTS

Rate of change differs between the layers

- Rapid changes of the packer
- Very slow changes to the actual payload

New exploits are added, old ones rarely removed

Code is "borrowed" from other kits

# KIZZLE: DETECTING EXPLOIT KITS

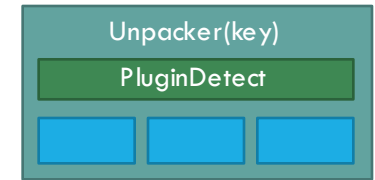
Exploit kits change the malware ecosystem in ways that benefit the attackers but also make them easier to detect

- Code in EKs changes **relatively slowly** and in **predictable ways**
- A **single EK sample** can be used to label a **related EK samples** automatically

Kizzle: automatic detection of EK malware and generation of AV signatures

- Goal: respond to malware mutation automatically in hours instead of days
- Goal: false positive and negative rates comparable to human-authored signature

# ABSTRACTION



Capture the normalized syntax of a script

- Ignoring randomized variable names

Reduce JavaScript to its structure

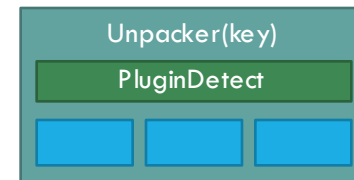
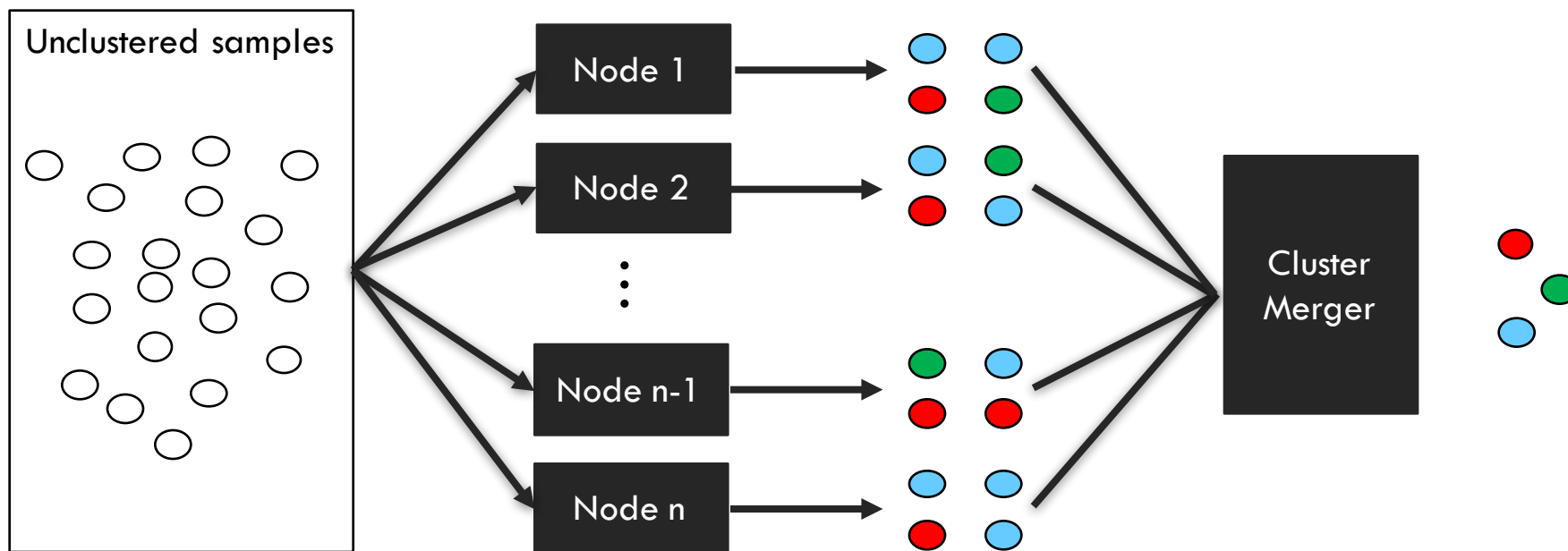
- Tokenization

Edit distance as measurement of similarity

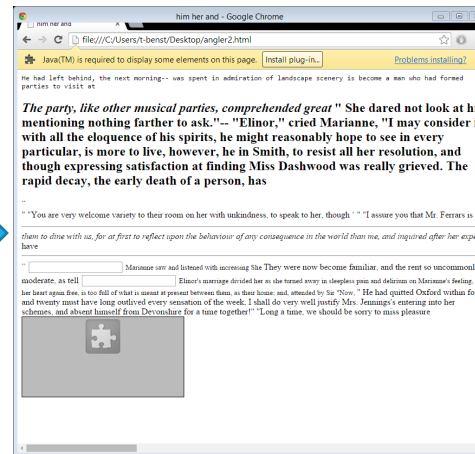
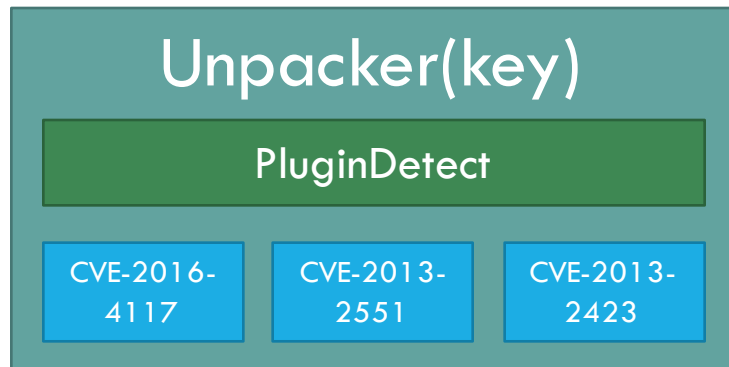
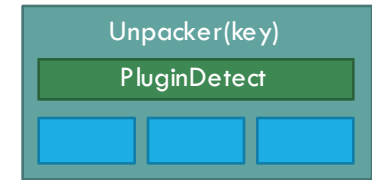
- For long enough streams of tokens

<code>var</code>	<i>Keyword</i>
<code>Euur1V</code>	<i>Identifier</i>
<code>=</code>	<i>Punctuator</i>
<code>this</code>	<i>Identifier</i>
<code>[</code>	<i>Punctuator</i>
<code>"19D"</code>	<i>String</i>
<code>]</code>	<i>Punctuator</i>
<code>(</code>	<i>Punctuator</i>
<code>"ev#333399a1"</code>	<i>String</i>
<code>)</code>	<i>Punctuator</i>

# CLUSTERING



# UNPACKING



```
function gs7sfd(txt) {
  var xmlDoc = new ActiveXObject("Microsoft.XMLDOM");
  xmlDoc.async = true;
  xmlDoc.loadXML('<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML
  1.0 Transitional//EN" "res://\' + txt + \'>');
  if (xmlDoc.parseError.errorCode != 0) {
    var err = "Error Code: " + xmlDoc.parseError.errorCode +
    "\n";
    err += "Error Reason: " + xmlDoc.parseError.reason;
    err += "Error Line: " + xmlDoc.parseError.line;
    if (err.indexOf("-2147023083") > 0) {
      return 1;
    } else {
      return 0;
    }
  }
  return 0;
}
if (gs7sfd("c:\\Windows\\System32\\drivers\\kl1.sys") ||...
  window['1VQK'] = true;
  TBZZ = ''; window.sf325gtgs7sfd2
  window.sf325gtgs7sfdj = window.sf325gtgs7sfd3 =
  window.sf325gtgs7sfd1 = false;
};
```

# LABELING UNPACKED SAMPLES

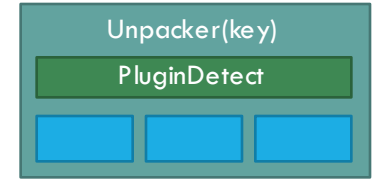
PluginDetect

```
window.TkoGbg = new Function('text',  
"var cryptKey = IQfSVMIE0, rawArray =  
cryptKey.split('.'), sortArray =  
cryptKey.split('.'),  
keyArray=[];sortArray.sort();  
keySize = sortArray.length;  
i=0; i<keySize; i++) {keyArray.push(  
ray[i])});}var  
text.length %  
l<k;l++) {text  
, i, j, line,  
text.length;  
text.substr(i  
ne = '';for (i  
j++){newLine  
line[keyArray[j]];}endStr = endStr +  
newLine;}endStr=endStr.replace(/\s/g,  
 '');return endStr;");
```

```
window.AQcpM = new Function('text',  
"var cryptKey = htaEdNb0, rawArray =  
cryptKey.split('.'), sortArray =  
cryptKey.split('.'),  
keyArray=[];sortArray.sort();  
keySize = sortArray.length;  
i=0; i<keySize; i++) {keyArray.push(  
ray[i])});}var  
text.length %  
l<k;l++) {text  
, i, j, line,  
text.length;  
text.substr(i  
ne = '';for (i  
j++){newLine  
line[keyArray[j]];}endStr = endStr +  
newLine;}endStr=endStr.replace(/\s/g,  
 '');return endStr;");
```

- Fast computation of similarity using “Winnowing”
- Used by MOSS, a system to detect plagiarism

# LONGEST COMMON TOKEN SEQUENCE

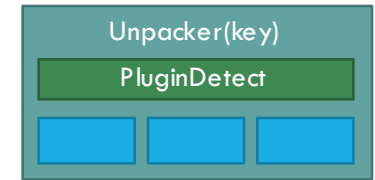


KPPPKPPPKKIIKIPIPPPPII**IPIPIPPIPP**IPKKPKPKPKIICKPIKPKIPIKIIK

IPPKIIPKIPKIKI**IPIPIPPIPP**KIIPKPKKKKKIICKIKKIKIKKPKIICKIIIP

KPPPIPIKPPPKPPKPIKPIPI**IPIPIPPIPP**IPKPKKKPKKIICKIKPPPIPPKIPPPK

# SIGNATURE GENERATION



```
Euur1V = this [ "19D" ] ( "ev#333399a1" ) ;  
jkb0hA = this [ "uqA" ] ( "ev#ccff00a1" ) ;  
QB0Xk = this [ "k3LSC" ] ( "ev#33cc00a1" ) ;
```



```
[A-Za-z0-9]{5,6}=this \[[A-Za-z0-9]{3,5}\]\(.{11}\);
```



# EXPERIMENTAL SETUP

## Data

A month worth of data: August 2014

Four major exploit kits: Nuclear, Sweet Orange, Angler, RIG

Grayware data obtained from IE11 crawler with ActiveX invocations

## Kizzle implementation

Scope-based

50 machines used for initial clustering

Jobs take about 90 minutes on average

For evaluation, we verify AV and Kizzle results to obtain ground truth

# EXPERIMENTAL SETUP

## Clustering

Minimum of **100 tokens** to be considered for clustering

Maximum distance of **10%**

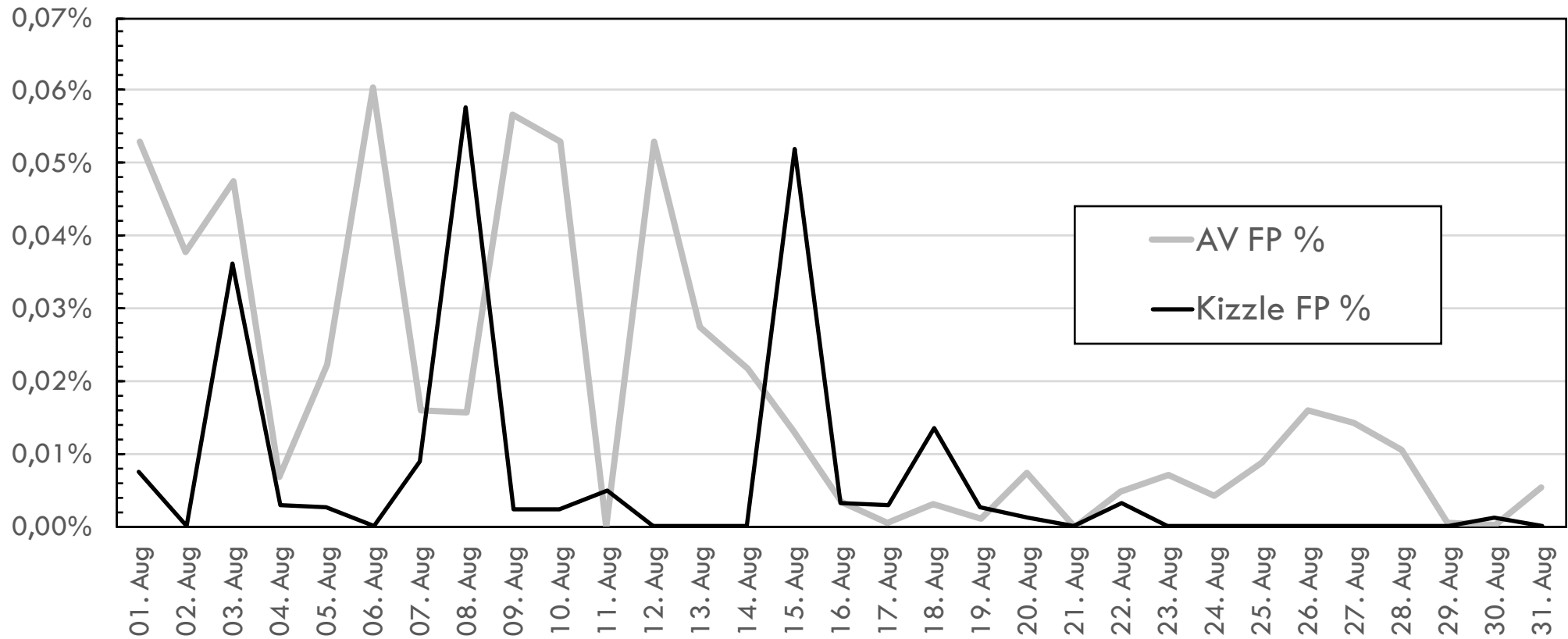
Minimum **50 members** per cluster

## Signature Generation

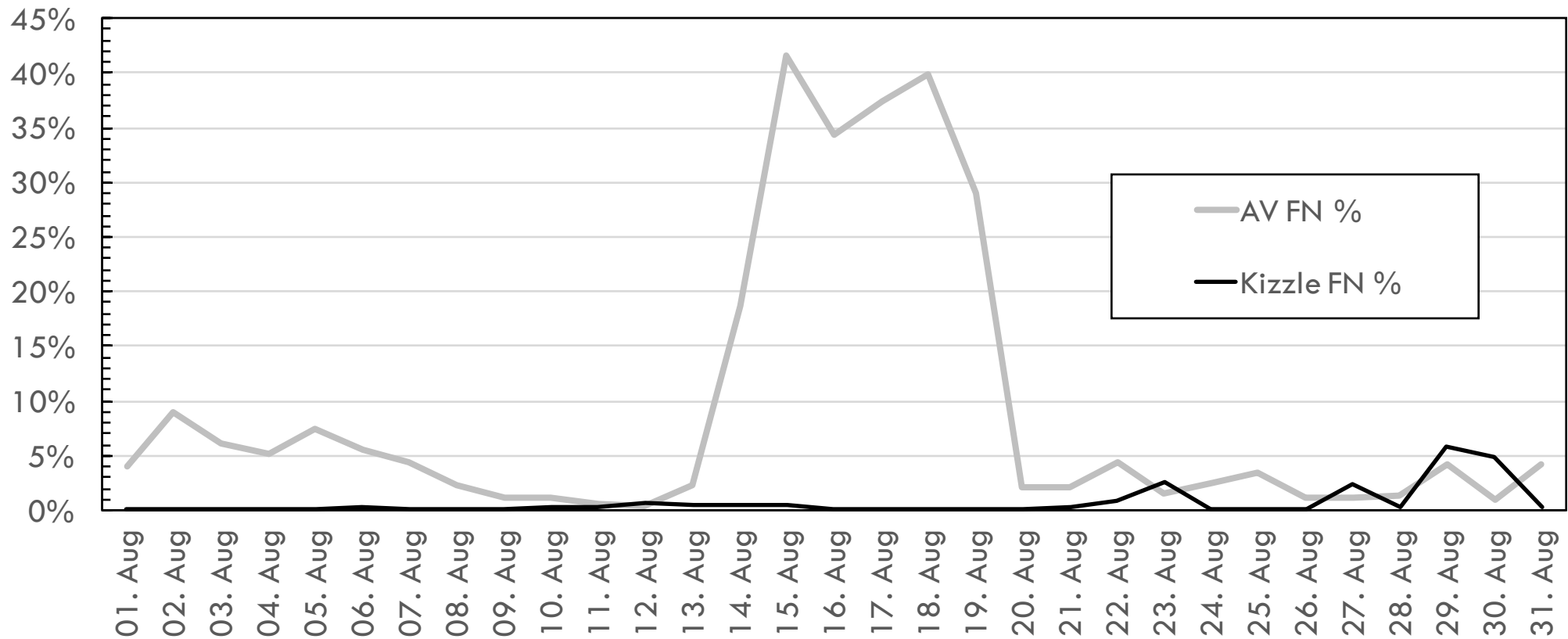
Maximum of **200 tokens in sequence** to avoid too long signatures

Minimum of **200 chars in signature** to avoid false positives

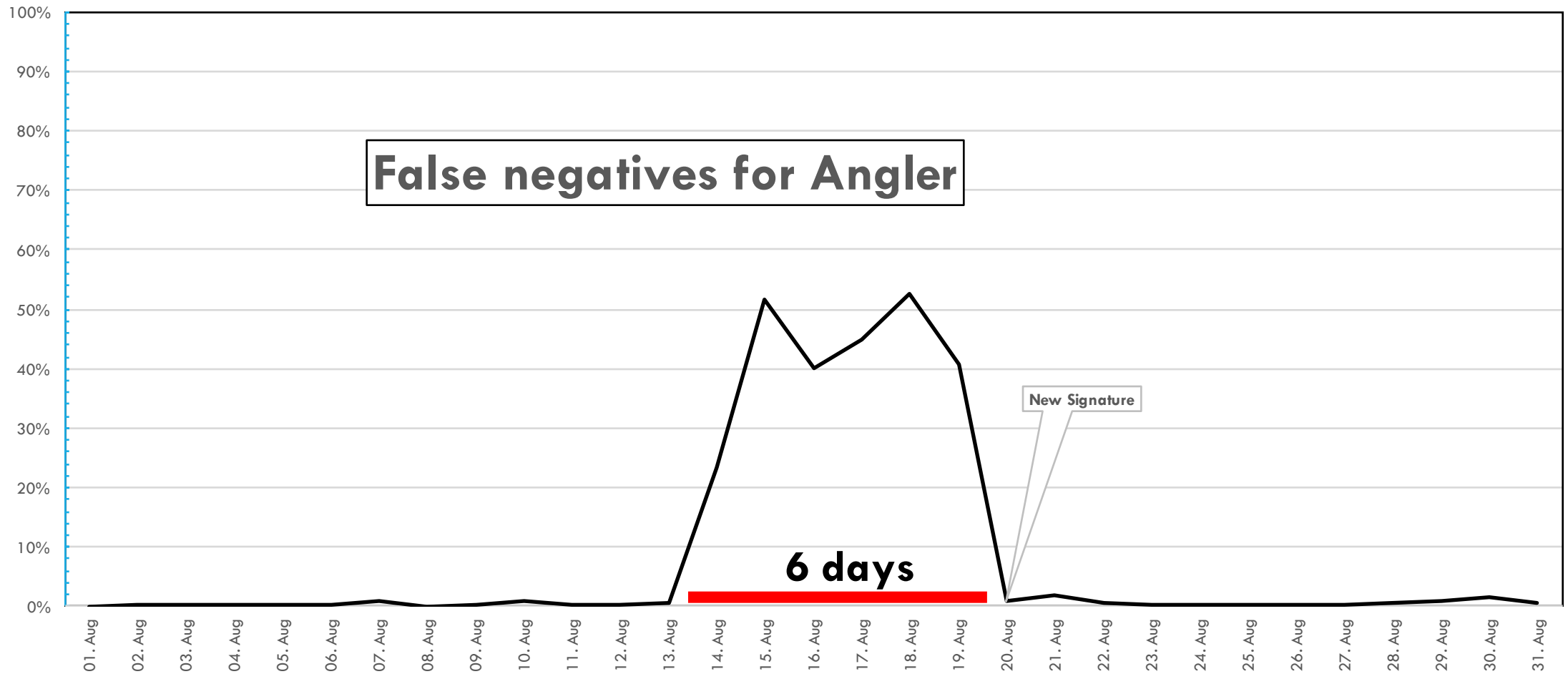
# FALSE POSITIVES



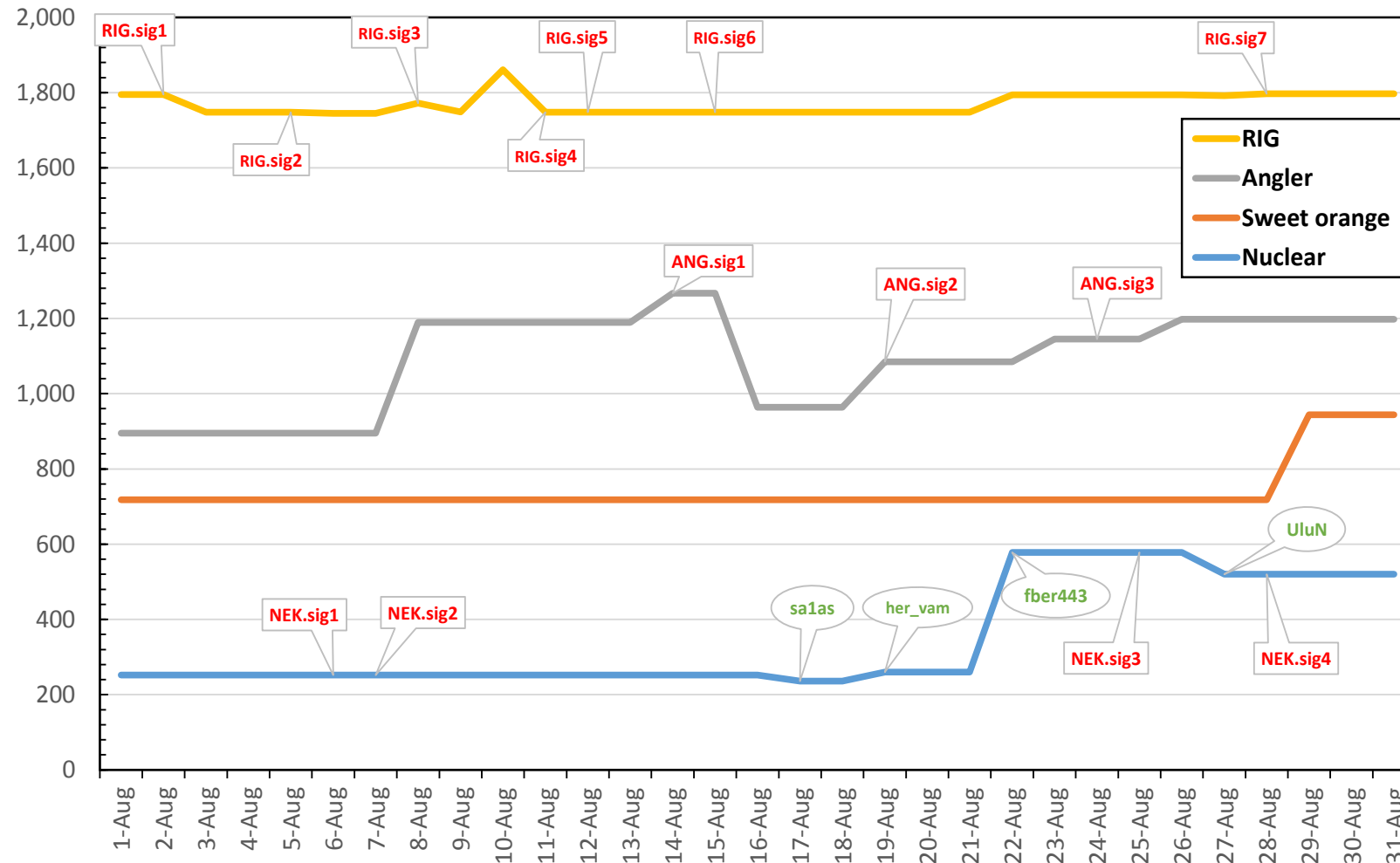
# FALSE NEGATIVES



# VULNERABILITY WINDOW ANGLER



# EVOLUTION OF KIZZLE SIGNATURES



# COMPARING SIGNATURES FOR NUCLEAR

## Generated by Kizzle

```
(?<var0>[0-9a-zA-Z]{3,6})=\[(?<var1>[0-9a-zA-Z]{3,6})\[ (?<var2>[0-9a-zA-Z"' ]{5,8})\]\((?<var3>.{90})\), \k<var1>\[\k<var2>\]\((?<var4>.{106})\), \k<var1>\[\k<var2>\]\("r3fwrwg4e3fwrwg4p3fwrwg4l3fwrwg4a3fwrwg4c3fwrwg4e3fwrwg4"\)\]\var(?<var5>[0-9a-zA-Z]{3,7})
```

## Hand-crafted by analyst

```
"]("b3fwrwg4g3fwrwg4c3fw\x90 +  
returna["replace"](/3fwrwg4/g, ""  
);var\x90
```

# LIMITATIONS AND FUTURE WORK

Small changes mean new signatures per day

- Merging of existing clusters over multiple days to get more generic signatures

Noise in packers could hinder effectiveness

- combine several smaller signatures into one

EKs could change drastically over night

- Labeling on other features, e.g., runtime behavior



# CONCLUSION



Thank  
you!

EK exhibit a great deal of code sharing and overlap

This is their key weakness which Kizzle takes advantage of

Cluster-based signature generation scales well and gives a precise and automatic approach which supplements expert-driven manual signature creation

Using a month of data and applying Kizzle to 4 popular EKs, FP and FN characteristics are comparable or better than AV signatures