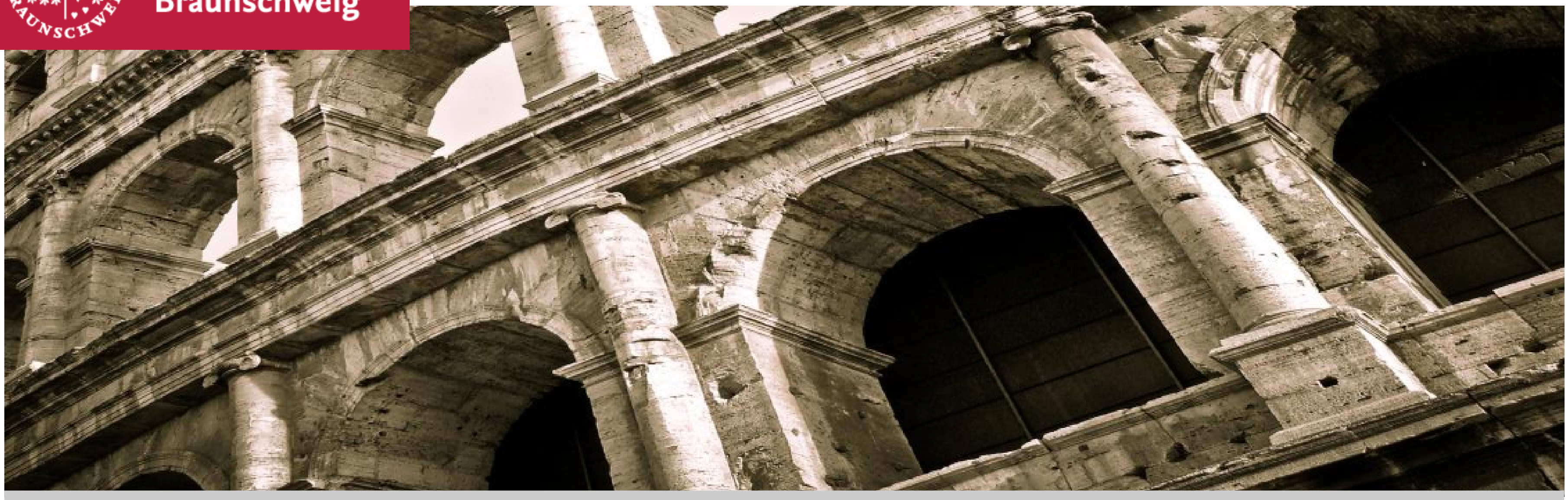
Technische Universität Braunschweig

Institute of System Security



Analyzing and Detecting Flash-based Malware

Christian Wressnegger, Fabian Yamaguchi,
Daniel Arp, and Konrad Rieck

ROMA MMXVI | APPSEC EUROPE

Christian Wre!§“$ADG=(„#$….who?

- **PhD candidate at the Institute of System Security**
  - Established in April 2016 in Brunswick, Germany by Prof. Konrad Rieck
  - *Previously at the University of Göttingen*
- **TU Braunschweig**
  - Oldest „institute of technology“ in Germany (founded in 1745)
  - 40-year-long history of computer science

Malware

- **Malicious software (Malware)**
  - Lasting problem of computer security
  - Omnipresence of Trojans, Bots, Adware, …
  - Increase of targeted attacks using Malware
- **Flash-based malware**
  - Malware targeting the Adobe Flash platform
  - Drive-by-Downloads, malicious redirects, exploits, …

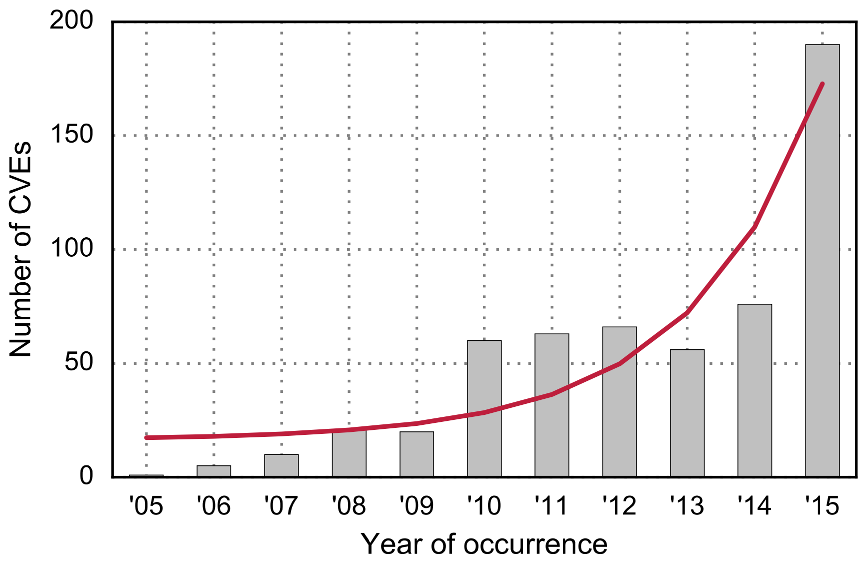
Adobe Flash

- **Flash is dead!**
  - Deployed on 500 million devices across different platforms
  - Used on 25% of the top 1,000 Alexa web sites
- **Dynamic and multimedia content on web pages**
  - *Advertisement, video streaming, gaming, …*
  - 20 years of deployment
  - Powerful scripting language: ActionScript

Adobe Flash Vulnerabilities

- **Increasing number of CVEs**
  - About 550 different vulnerabilities in total
  - Until 2015: 167 new vulnerabilities (80% code execution)

Disclaimer! Effective August 2015



Attack Vectors and Scenarios

1. **Structural Exploits against the Flash Player**
   - Vulnerabilities in the file format parser
2. **Malicious ActionScript code**
   - Launching or preparing exploits *(Obfuscation, heap-spraying, …)*
3. **Environment fingerprinting**
   - Selecting targets based on interpreter or OS information

Concrete attacks may fall into more than one of these categories

Obfuscation

- **Staged execution**
  - Dynamic code-loading in form of another animation
    loadMovie (ActionScript 2), Loader object (ActionScript 3)
  - Layered encryption/ polymorphism
    Runtime-packers (secureSWF, DoSWF)
  - Exploit legacy code
- **Source-code Obfuscation**
  - Variable substitution, string assembly, dead code, etc.
- **Probing the execution environment**
  - Triggering a malware's payload on specific systems only

Probing the environment

- **Information about the execution environment**
  - System.capabilities (ActionScript 2)
  - flash.system.Capabilities (ActionScript 3)
- **LadyBoyle malware exploiting CVE-2015-323**

```
switch (this.version) {
    case "win 11,5,502,146": break;
    case "win 11,5,502,135": break;
    case "win 11,5,502,110": break;
    case "win 11,4,402,287": break;
    case "win 11,4,402,278": break;
    case "win 11,4,402,265": break;
    default:
        return this.empty();
}
```

md5: cac794adea27aa54f2e5ac3151050845

- **Comprehensive analysis of Flash animations**
  Support for all versions of ActionScript and Adobe Flash platforms
  - Structural Analysis *(static)*
  - Guided code-execution *(dynamic)*
- **Learning-based detection of Flash-based malware**
  - Detects 90–95% of malicious Flash files at 0.1% and 1.0% FPs
    - Significantly outperforms related approaches
    - Best learning-based detector for Flash-based Malware
  - No need for manually constructed detection rules

Structural Analysis

- **Flash animations are composed out of "tags"**
  - Containers to store code, animation specs and data *(audio, video, images, fonts, etc.)*
  - Future versions may extend on the number of tags
  - Possible occurring nested *(DefineShape, ...)*
- **Offering a huge attack surface**
  - Many exploits rely on a specific (sequences of) tag
  - Memory corruption exploits such as CVE-2007-0071

Structure Reports

- **Exemplary report for a LadyBoyle sample using** CVE-2015-323

```
69   FileAttributes
77   Metadata
 9   SetBackgroundColor
 2   DefineShape
     39   DefineSprite
     26   PlaceObject2
86   DefineSceneAndFrameLabelData
43   FrameLabel
87   DefineBinaryData  // Payload
87   DefineBinaryData  // Payload
82   DoABC             // ActionScript 3
76   SymbolClass
 1   ShowFrame
```

- **More compact:** `69 77 9 2 [ 39 26 ] 86 43 87 87 82 76 1`

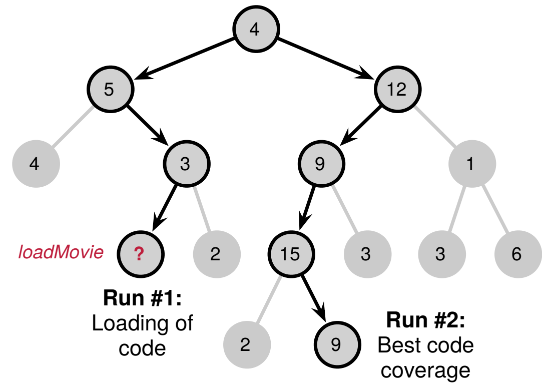
md5: cac794adea27aa54f2e5ac3151050845

Analyzing Code

- **Dynamic code analysis**
  - Single execution “as-is” is not sufficient
  - Covering all execution paths is not feasible
  - Heuristics needed!
- **Previous approaches**
  - Determine which paths to execute based on external input
    (“Exploring Multiple Execution Paths for Malware Analysis“, Moser et al.)
  - Symbolic execution of code
    (“A Symbolic Execution Framework for JavaScript“, Saxena et al.)
  - Multi-execution of branches along the intended path
    (“Rozzle: De-cloaking Internet Malware“, Kolbitsch et al.)

Guided Code-Execution

- **Gordon:** Guide the interpreter towards indicative code regions
  - Branches that contains indicative functions
    (loadMovie, loadBytes, ByteArray, ...)
  - Paths with many instructions
- **Two-step procedure**
  - Determine Control-flow statically
  - Use CFG to guide the analyzer
    - Multiple runs possible
    - Force Execution at environment sensitive conditions



Execution Reports

- **Excerpt of a report for a sample using** CVE-2015-323

```
R1 973:  pushString   "fla"
R1 975:  pushString   "sh.uti"
R1 977:  add          "fla" + "sh.uti"
R1 978:  pushString   "ls.Byt"
R1 980:  add          "flash.uti" + "ls.Byt"
R1 981:  pushString   "eArray"
R1 983:  add          "flash.utils.Byt" + "eArray"
R1 984:  callProperty [ns:flash.utils] getDefinitionByName 1
R1 >     Looking for definition of
R1 >         [ns:flash.utils] ByteArray
R1 >     Getting definition for
R1 >         [ns:flash.utils] ByteArray
R1 987:  getLex: [ns:] Class
```

- For automatic processing reports meta data is omitted

md5: 4f293f0bda8f851525f28466882125b7

Learning-based Detection

- **Preprocessing of reports**
  - Structure reports: cf. compact representation

    ```
    69 77 9 2 [ 39 26 ] 86 43 87 87 82 76 1
    ```

  - Execution reports: Instruction names and parameters only
    - Parameters are replaced with their respective type

    ```
    pushString   STR
    add          STR + STR
    callProperty getDefinitionByName NUM
    getLex       ID
    ```

- **Embedding:** n-gram models of structure and execution reports
- **Learning:** Classification using Support Vector Machines *(SVMs)*

n-Gram Models

- **Used to embed string data into vector space**
  - Generalization of the Bag-of-Words model
  - String represented as bag of features
- **Different variations:**
  - Words
  - Byte n-grams
  - Word n-grams

n-Gram Models

- **Used to embed string data into vector space**
  - Generalization of the Bag-of-Words model
  - String represented as bag of features
- **Different variations:**
  - Words
  - Byte n-grams
  - Word n-grams

Used	data
to	into
embed	vector
string	space

n-Gram Models

- **Used to embed string data into vector space**
  - Generalization of the Bag-of-Words model
  - String represented as bag of features
- **Different variations:**
  - Words
  - Byte **5** grams
  - Word n-grams

Used␣
sed␣t
ed␣to
d␣to␣
␣to␣e

to␣em
o␣emb
␣embe
embed
...

n-Gram Models

- **Used to embed string data into vector space**
  - Generalization of the Bag-of-Words model
  - String represented as bag of features
- **Different variations:**
  - Words
  - Byte n-grams
  - Word **3** grams

Used to embed

to embed string

embed string data

string data into

...

Embedding of n-grams in Vectors

- **Assign each n-gram a dimension in the vector**

$$\Phi : x \longrightarrow \left(\Phi_s(x)\right)_{s \in S}$$

- **Embeddings**
  - Counting $\quad \Phi_s = \quad \#\ n\text{-gram } s \text{ in } x$
  - Binary occurrence $\quad \Phi_s = \begin{cases} 1 & n\text{-gram } s \text{ in } x \\ 0 & \sim \end{cases}$

n-Grams of Tag Identifiers

- **Example for the structure report**
  - 4-grams of tag identifiers

69	77	9	2	**[**	39	26	**]**	86	43	87	87	82	76	1

69	77	9	2				]	86	43	87				
	77	9	2	[				86	43	87	87			
		9	2	[	39				43	87	87	82		
			2	[	39	26				87	87	82	76	
				[	39	26	]				87	82	76	1

n-Grams of Instructions and Parameters

- **Example for the execution report**
  - 4-grams of instructions/ params

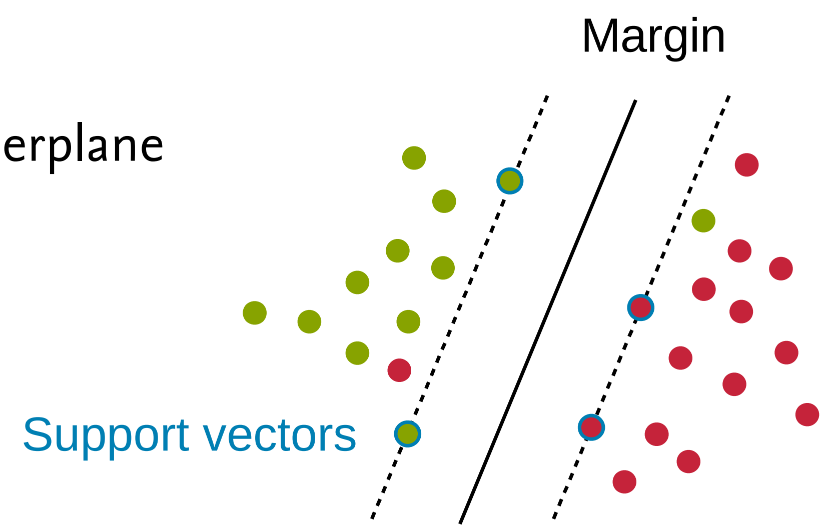
```
pushString STR add STR
           STR add STR +
               add STR + STR
                   STR + STR callProperty
                         + STR callProperty getDefinitionByName
```

...

- **No need for manually constructing detection rules**
  - Implicit representation of instruction counts, call frequencies, etc.

Learning the Classifier

- **Support Vector Machines (SVMs)**
  - Modern supervised learning algorithm for classification
  - Invented by Vapnik *(1963)* and kernelized by Boser *(1992)*
  - Well-known for its effectiveness, efficiency and robustness
- **Important concepts**
  - Hyperplane with maximum margin
  - Regularization by softening the hyperplane
    - Let's you compensate mistakes



Evaluation

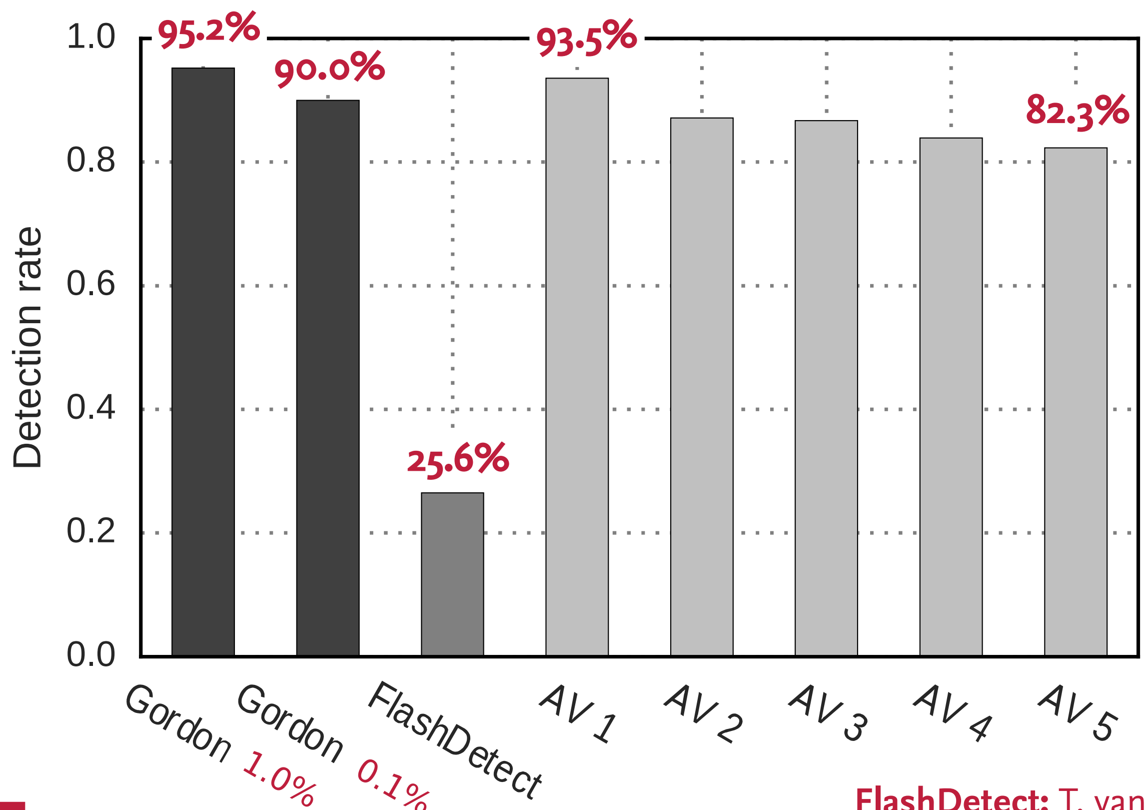
- **Datasets**
  - 26,600 Flash Animations collected over 12 weeks
  - 1,923 malicious and 24,671 benign samples
- **How well are we able to detect Flash-based malware?**
  - Comparison to the state-of-the-art methods
  - Is Gordon applicable in a continuous setting?
- **What's all the fuss about two different analyses?**
  - Wouldn't be one of them enough?

Experimental Setting

- **Temporal split of the data**
  - Weeks 1-6 for training, weeks 7-9 for validation, and the remainder, weeks 10-12 for testing
  - All test data has been collected after training
- **Related approaches**
  - FlashDetect *(T. van Overveldt et al, RAID 2012)*
    - Adjusted to 1% false-positives
    - Not supported version have been excluded (version 8 and below)
  - Virus scanners listed at VirusTotal

Comparative Evaluation

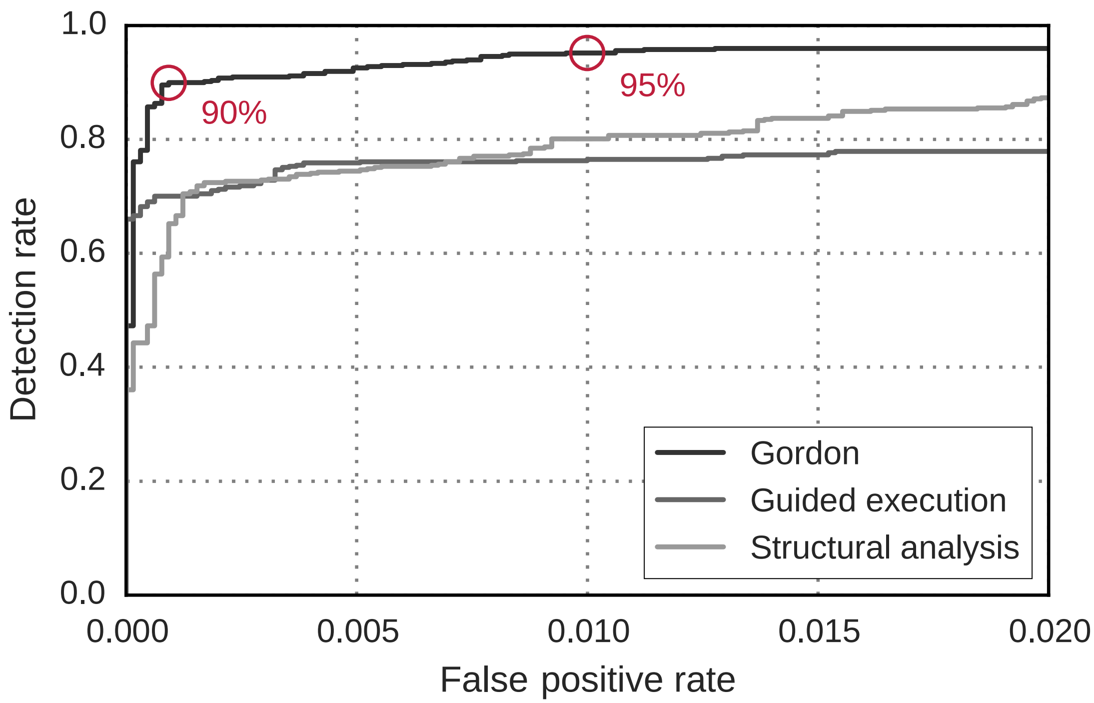
- **Gordon is on a par with tradition approaches**
  - No manual effort needed, though



FlashDetect: T. van Overveldt et al, RAID 2012

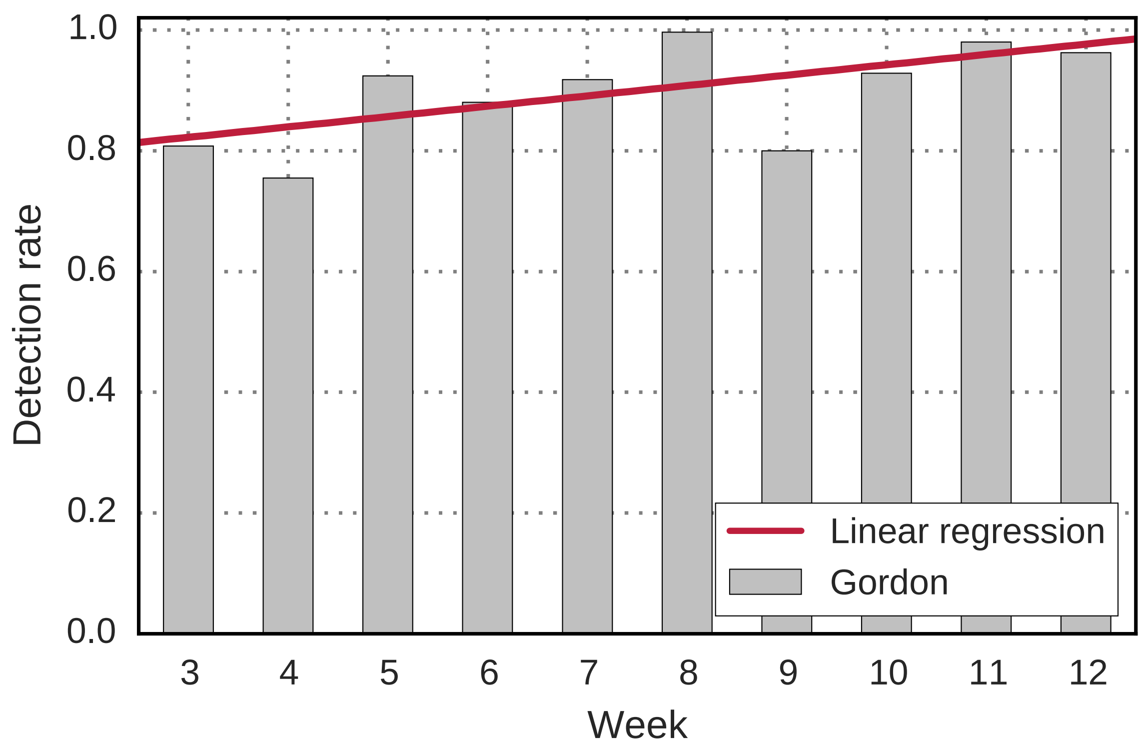
Combined Detection Performance

- **Gordon benefits from two orthogonal analyses**
  - Individual representations only detect 60–65% at 0.1% FPs



Temporal Evaluation

- **Applied to 12 consecutive weeks:** 80–99% detection rate
  - Clear trend towards Gordon's optimal performance



Summary

- **Comprehensive Analysis of Flash-based malware**
  - Structural analysis
  - Guided code-execution
    - Directed analysis of indicative code regions
- **Effective Detection of a large variaty of Flash-based malware**
  - High detection rate: 90–95% of malicious samples
    - Low false-positve rates
    - Best learning-based detector for Flash-based Malware
  - Can be used to bootstrap traditional methods

Thank you. Questions?