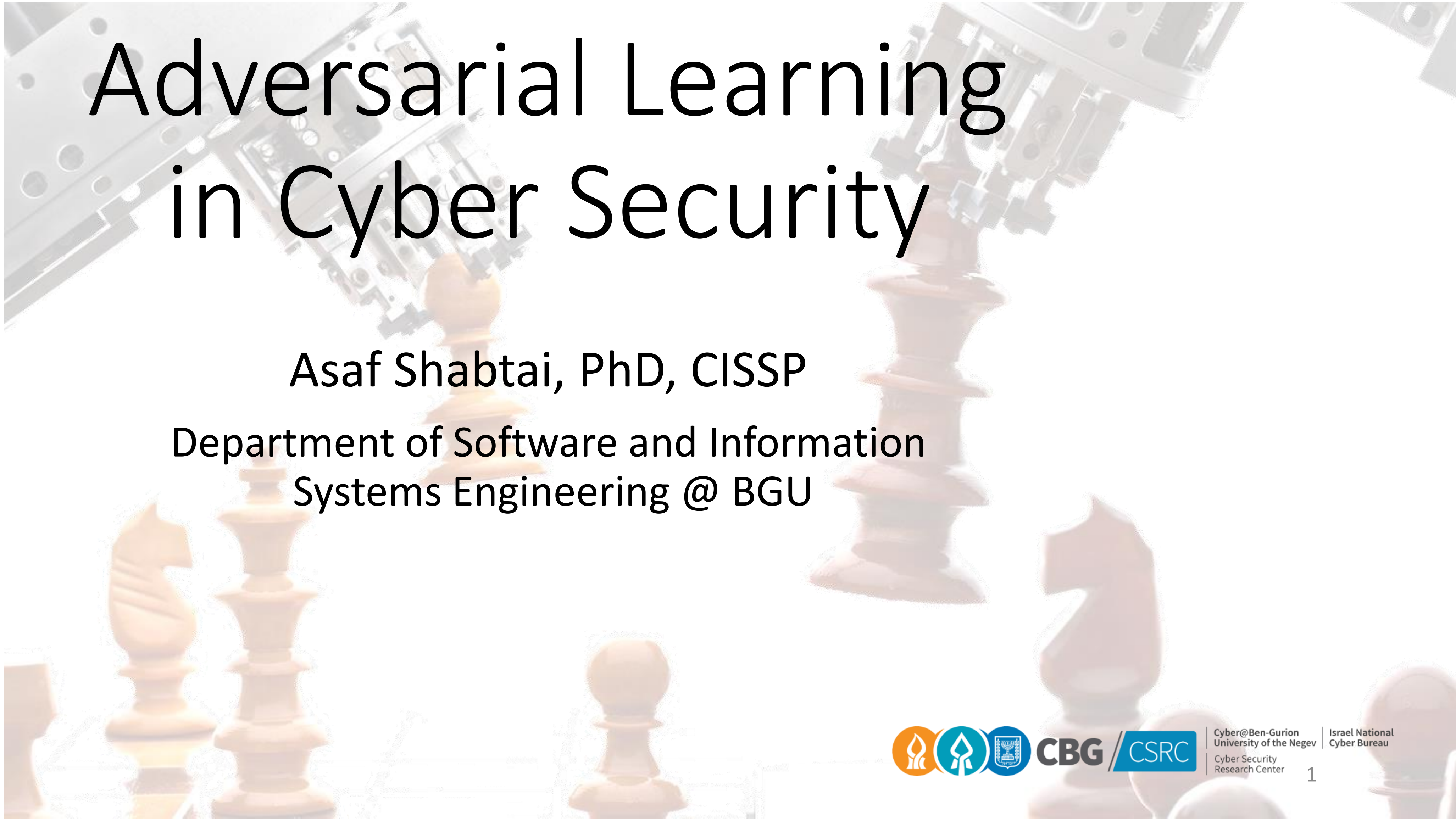


Adversarial Learning in Cyber Security

Asaf Shabtai, PhD, CISSP

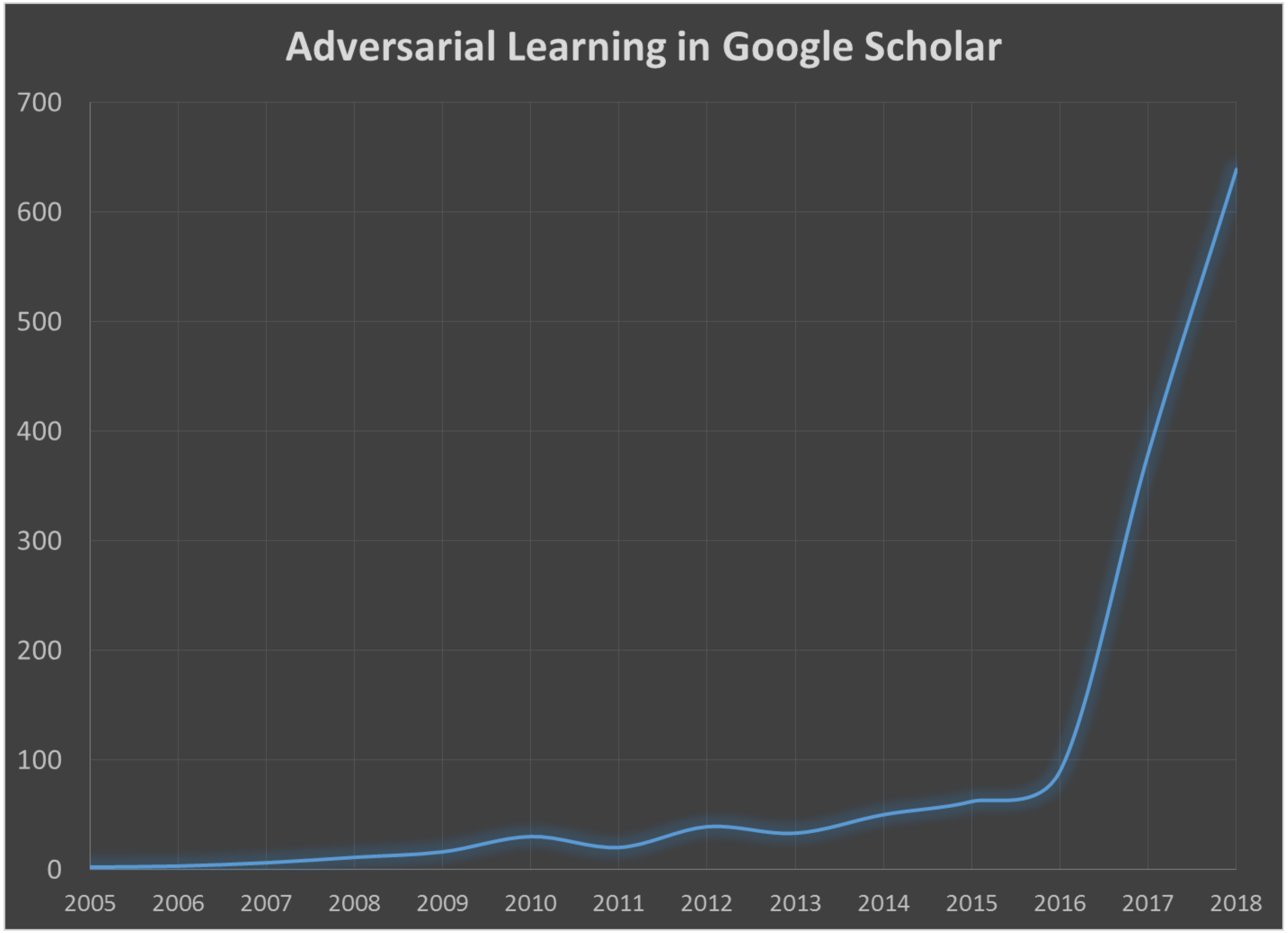
Department of Software and Information Systems Engineering @ BGU

CBG / CSRC
Cyber@Ben-Gurion University of the Negev
Cyber Security Research Center
Israel National Cyber Bureau

Adversarial learning

- Due to the large increase in the use of AI methods, hackers concluded that they too should embrace AI
- Today, hackers are using machine learning to find loopholes in other machine learning based systems
- Fooling AI systems is not very hard as machine learning relies on past cases and assumes that future data shares its characteristics
- Hackers abuse this assumption for example, by manipulating the input data

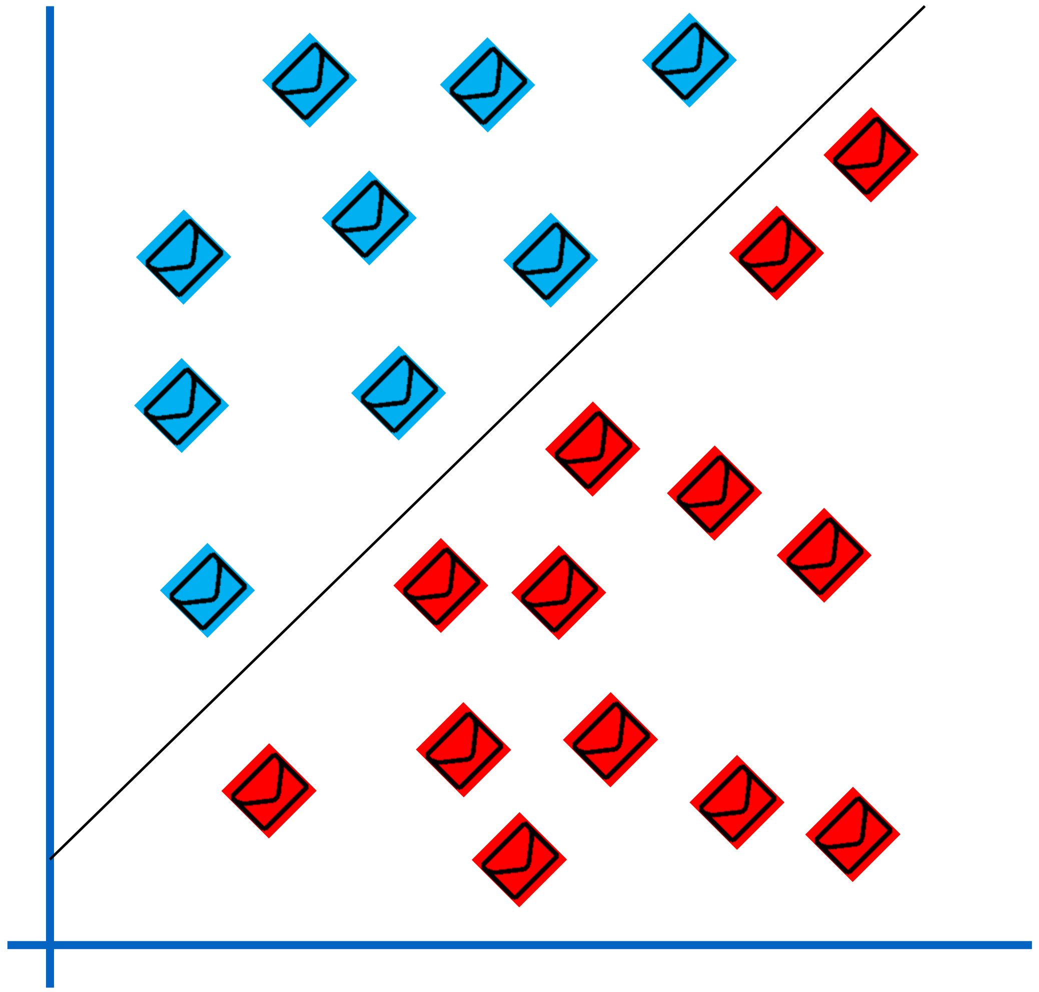
Academic research on AL



Spam detection system

Trained to look for **incriminating content** by analyzing the text of spam emails

To avoid detection, a spammer can obfuscate the content of an email by **deliberately misspelling suspicious words**



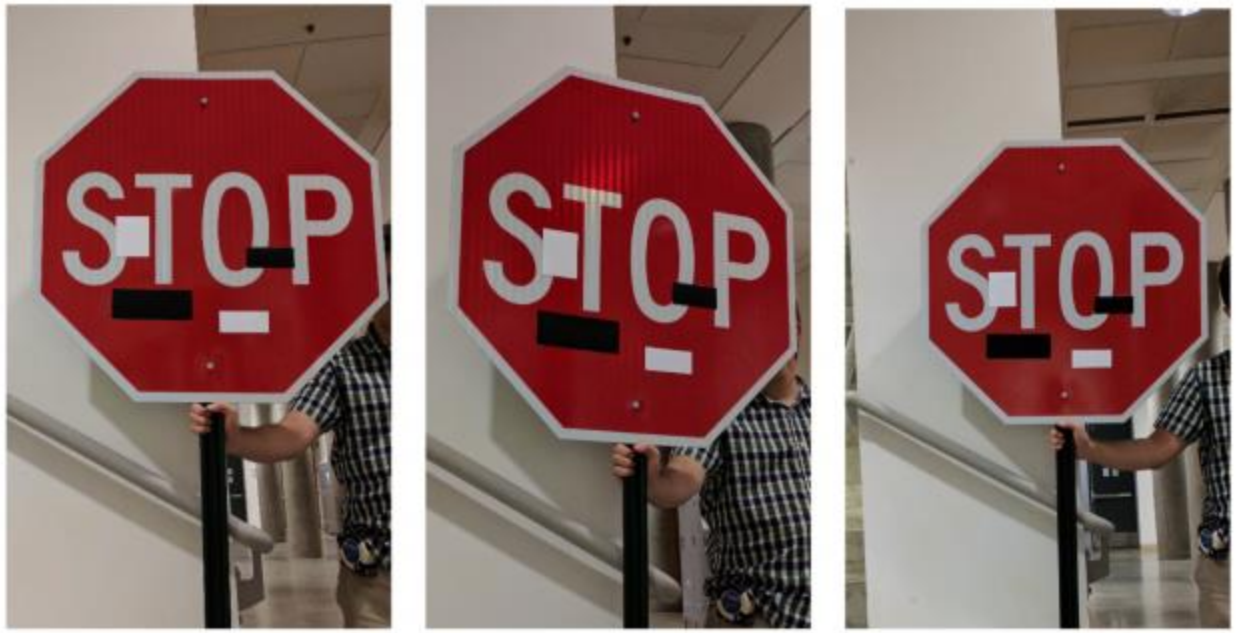
Adversarial traffic signs

Robust Physical-World Attacks on Deep Learning Visual Classification

Use **Deep Learning (CNN)** for image recognition (traffic signs identification) by **autonomous vehicles**

Adversarial AI can be also applied on machine vision systems

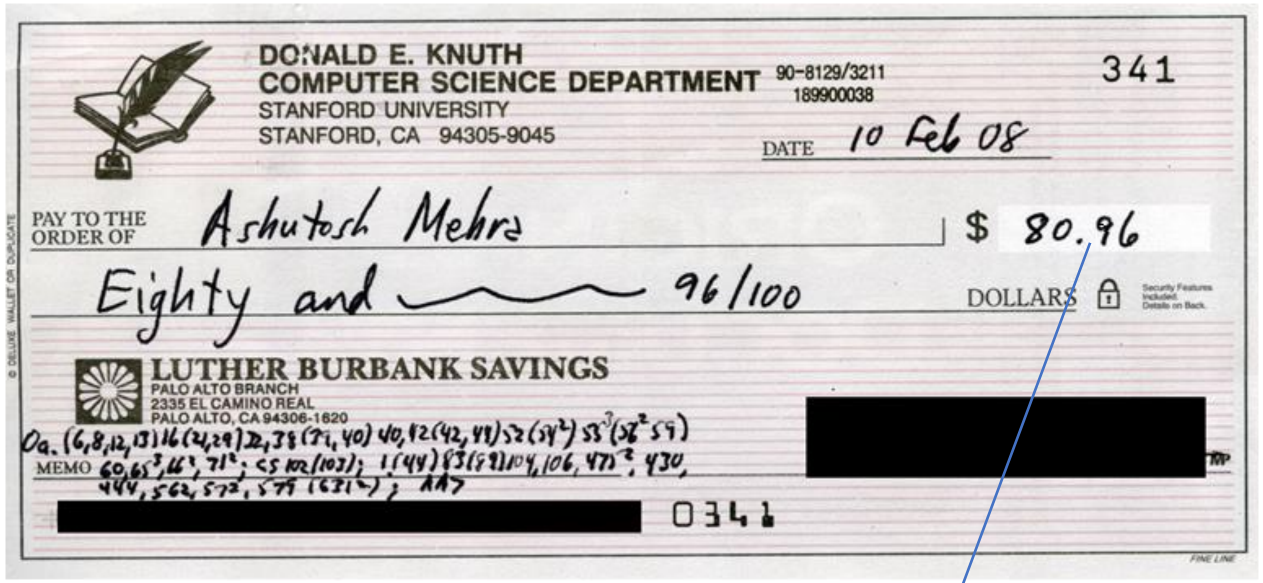
“**Confused the computer vision system** into thinking that a stop sign was a 45 mph sign, with just a few pieces of tape.”



Handwrite recognition system

If the presented check is examined by a handwriting recognition system, the amount is extracted correctly

However, by adding certain adversarial noise to the digit nine we can fool the system and make it think that it is the digit eight.



DONALD E. KNUTH
COMPUTER SCIENCE DEPARTMENT 90-8129/3211 189900038
STANFORD UNIVERSITY
STANFORD, CA 94305-9045

341

DATE 10 Feb 08

PAY TO THE ORDER OF Ashutosh Mehra $ 80.96

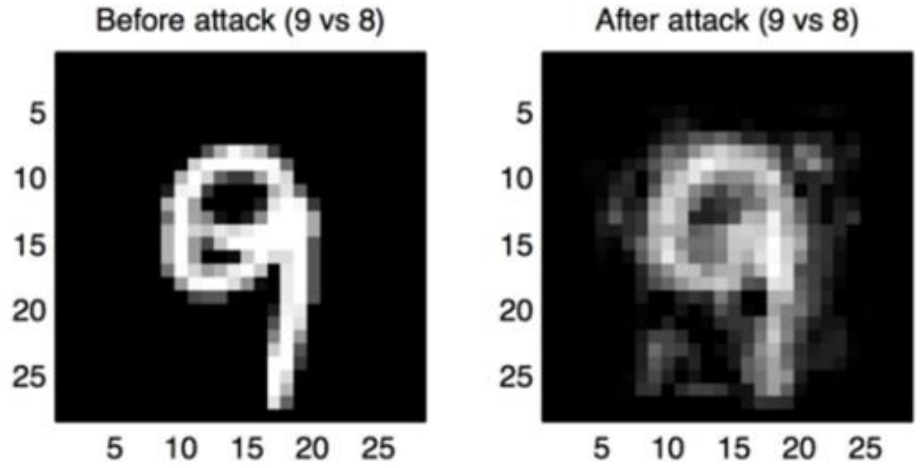
Eighty and 96/100 DOLLARS

LUTHER BURBANK SAVINGS
PALO ALTO BRANCH
2335 EL CAMINO REAL
PALO ALTO, CA 94306-1620

[redacted]

MEMO 0a. (6,8,12,13) 16 (24,29) 32, 38 (39, 40) 40, 42 (42, 48) 52 (54²) 55³ (56² 59) 60, 65³, 66², 71²; 85 102 (103); 1 (44) 83 (89) 104, 106, 471²; 430, 444, 562, 572, 579 (631²); 447

[redacted] 0341



Man vs. Machine: Practical Adversarial Detection of Malicious Crowdsourcing Workers [Wang et al. 2014]

Machine learning is being used for detecting fake/malicious profiles in social networks; analyzing content and structural features of graphs

It was shown how an attacker can evade detection

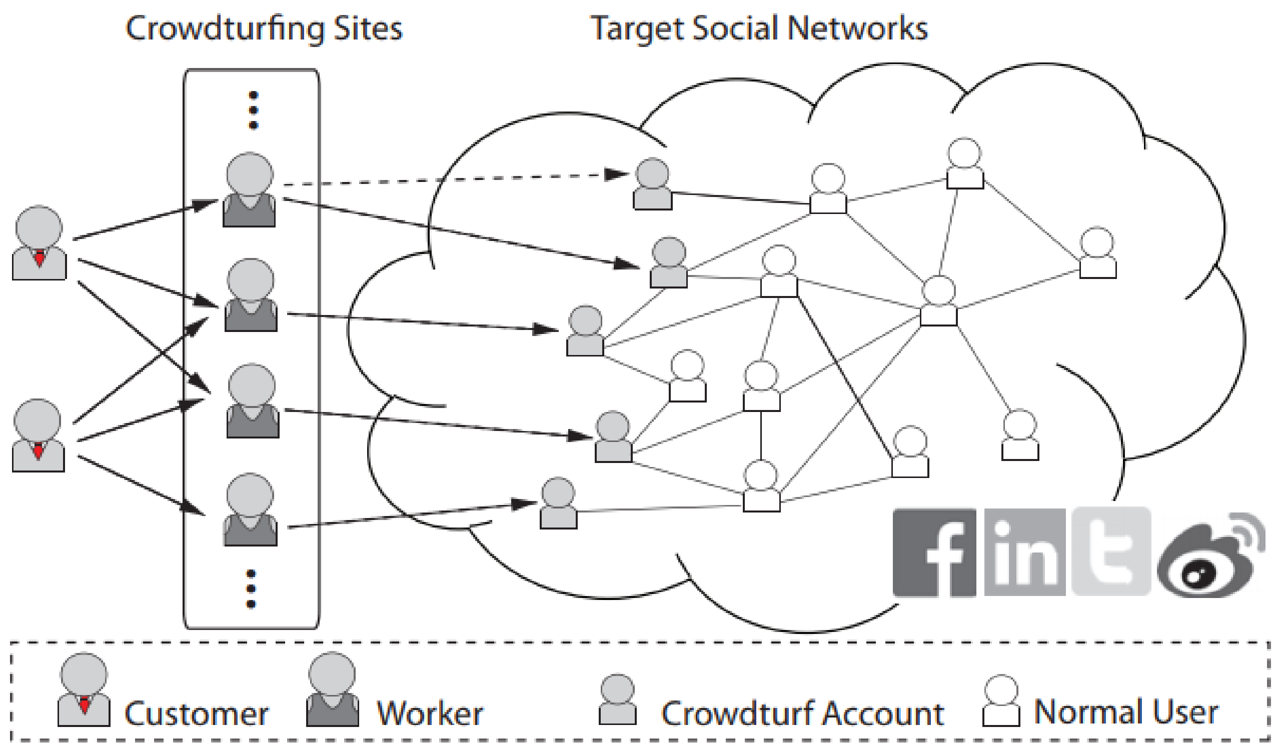


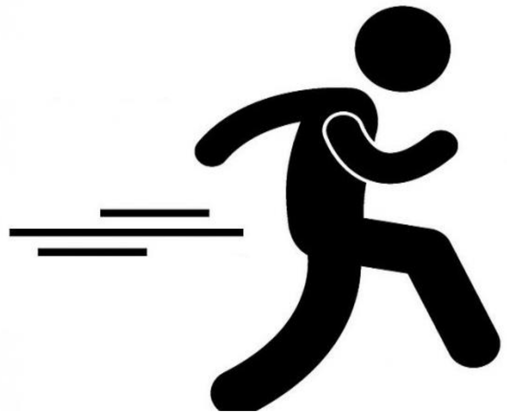
Figure 1: Crowdturfing process.

Categorization of adversarial attacks: **goals**

- **Confidentiality**: attempt to expose the model structure or parameters (IP) or the data used to train it (e.g., patient data)
  - Query the model
  - Performing membership test (to know whether an individual is in a training set)
  - use the model to complete an input vector with the most likely missing bits

Categorization of adversarial attacks: **goals**

- **Integrity**: attempt to control model's output by modifying
  - the input to the model (also known as evasion attack)
  - the training data (also known as poisoning attack)



Categorization of adversarial attacks: **goals**

- **Availability**: attempt to reduce
  - quality (e.g., confidence or consistency)
  - performance (e.g., speed)
  - access (e.g., denial of service)
    - Producing vision inputs that force a autonomous vehicle to stop
    - Allergy attack – Generate DoS attacks which bear many of the feature of regular traffic, inducing the IDS to block a lot of legitimate traffic [Chung, 2006]

Categorization of adversarial attacks: **attacker’s pre-knowledge**

- **White-box**: the adversary has some information about the model (type, parameters, architecture) or its original training data (summary, partial, or full training data)

- Adversarial example crafting [Szegedy, 2014]: access to the model and its parameters → identification of feature space parts for which the model has high error → altering an input to into that space

- Not always unrealistic
  - ML models deployed to smartphones [Hinton, 2014] in which case reverse engineering may enable adversaries to gain white-box access
  - Staged attacks - get the model first, than attack it directly [Rozenberg, 2016]

Categorization of adversarial attacks: **attacker's pre-knowledge**

- **Black-box**: the adversary has no knowledge about the model
  - still capable of issuing queries to the model or collecting a surrogate training dataset
- Oracle model - the adversary may issue queries to the ML model and observe its output for any chosen input [Papernot, 2017]
  - In ML as a Service cloud platforms, the model is potentially accessible through a query interface
  - Non-cloud APIs, e.g. the IOfficeAntivirus COM interface [Hamlen, 2009]
- In the network intrusion detection scenario:
  - white-box adversary has access to the model used to distinguish attacks from normal behavior
  - black-box adversary would only have access to TCP dumps of the network traffic

Categorization of adversarial attacks: **attack phase**

- **Training**: attempt to learn, influence or corrupt the model itself
  - read - simply accessing a summary, partial or all of the training data
  - injection - inserting adversarial inputs into the existing training data
  - modification - altering existing training data
  - logic corruption - tamper with the learning algorithm

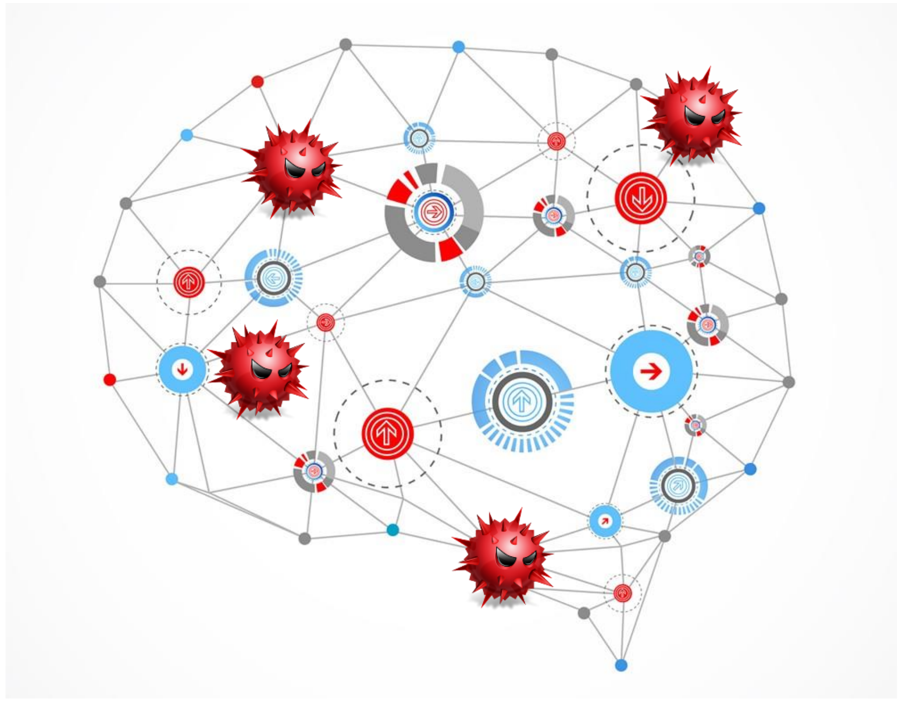
- **Operational**: attempt to interfere with the normal operation of the induced model
  - inference
  - evade
  - “steal the model”
  - denial-of-service

Categorization of adversarial attacks: **scope**

- **Targeted attack:** the focus is on a single or small set of target points (class)
- **Indiscriminate attack:** involves a very general class of points, such as “any false negative”

Generic Black-Box Attack Against API-Calls Based Malware Classifiers

Ishai Rosenberg, Yuval Elovici,
Lior Rokach, Asaf Shabtai



Malware detection using machine learning

Static analysis

- leverage structural information (sequence of bytes, strings, headers, functions)
- attempts to detect malware before the program is executed
- information about the program or its expected behavior consists of explicit and implicit observations in its binary/source code

Dynamic analysis

- leverage runtime information (network usage, files and memory modifications, system calls)
- attempts to detect malicious behavior during program execution

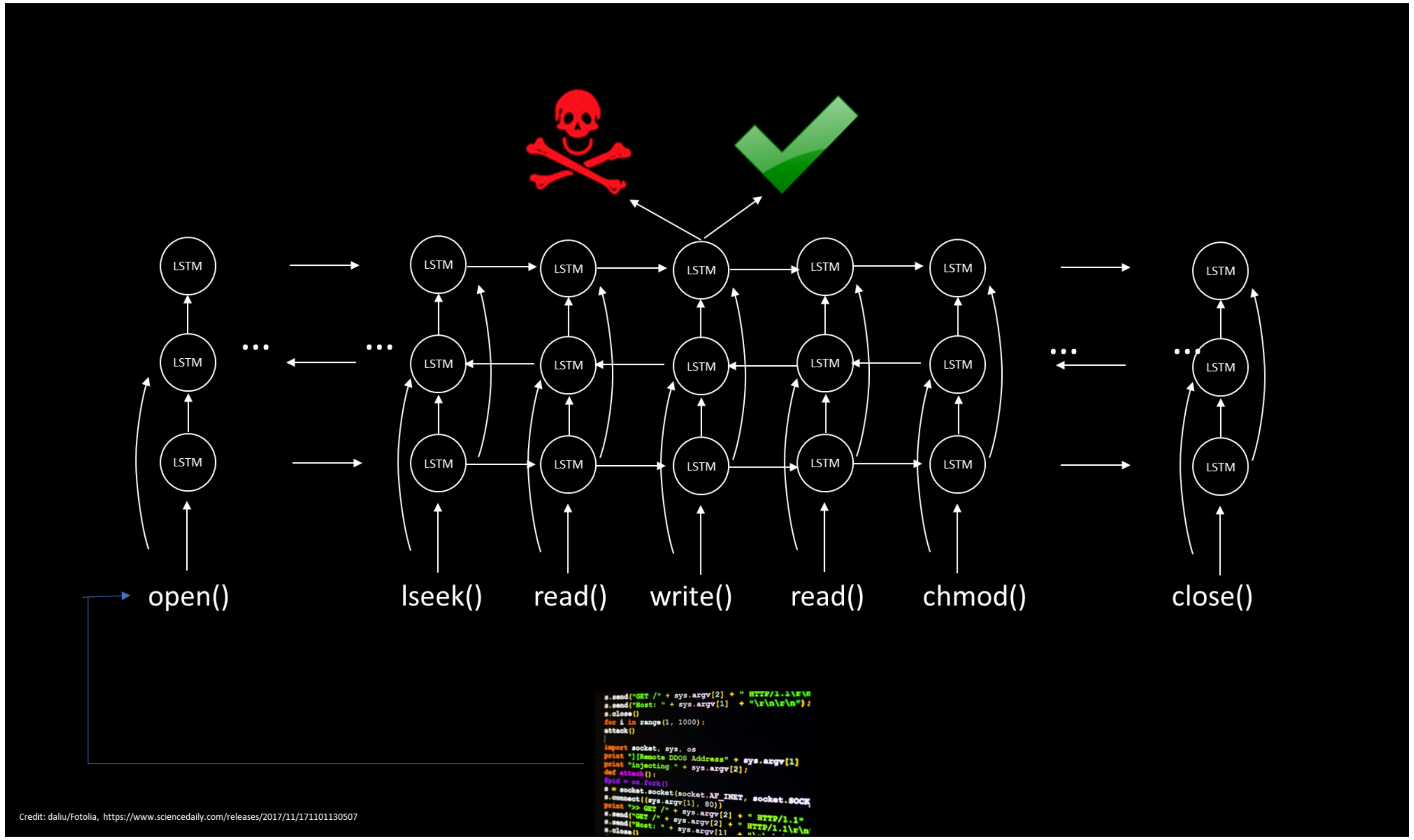
Recent **academic research** as well as **next generation anti-malware products** use **machine and deep learning** models instead of signatures and heuristics;
in this research we show how **such solutions can be evaded successfully**

Detecting malware by analyzing system calls

Many of the AI-based anti-viruses examine the sequence of system calls (extracted at runtime) to characterize an application's behavior

The sequence consists of requests issued by an application towards the operating system

Sequences used as features
- Classification based on extracted syscall n-grams
- Markov model
- ANN – RNNs [Pascanu et al., 2015]; RNN, LSTM, GRU and CNN [Athiwaratkun et al., 2017]; RNN, combining file access and API calls [Wang et al., 2016]; feed-forward DNN, combining static features and API calls [Huang et al., 2016]
- …

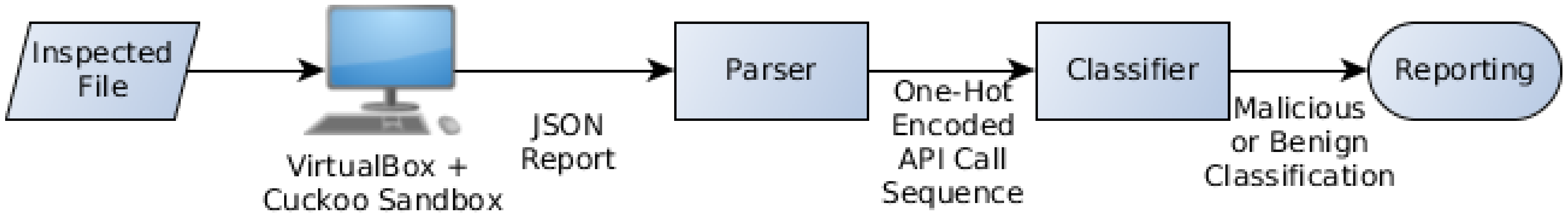


```
s.send("GET /" + sys.argv[2] + " HTTP/1.1\r\n
s.send("Host: " + sys.argv[1]  + "\r\n\r\n");
s.close()
for i in range(1, 1000):
attack()

import socket, sys, os
print "][Remote DDOS Address" + sys.argv[1]
print "injecting " + sys.argv[2];
def attack():
#pid = os.fork()
s = socket.socket(socket.AF_INET, socket.SOCK
s.connect((sys.argv[1], 80))
print ">> GET /" + sys.argv[2] + " HTTP/1.1"
s.send("GET /" + sys.argv[2] + " HTTP/1.1\r\n
s.send("Host: " + sys.argv[1]
s.close()
```

Credit: daliu/Fotolia, https://www.sciencedaily.com/releases/2017/11/171101130507

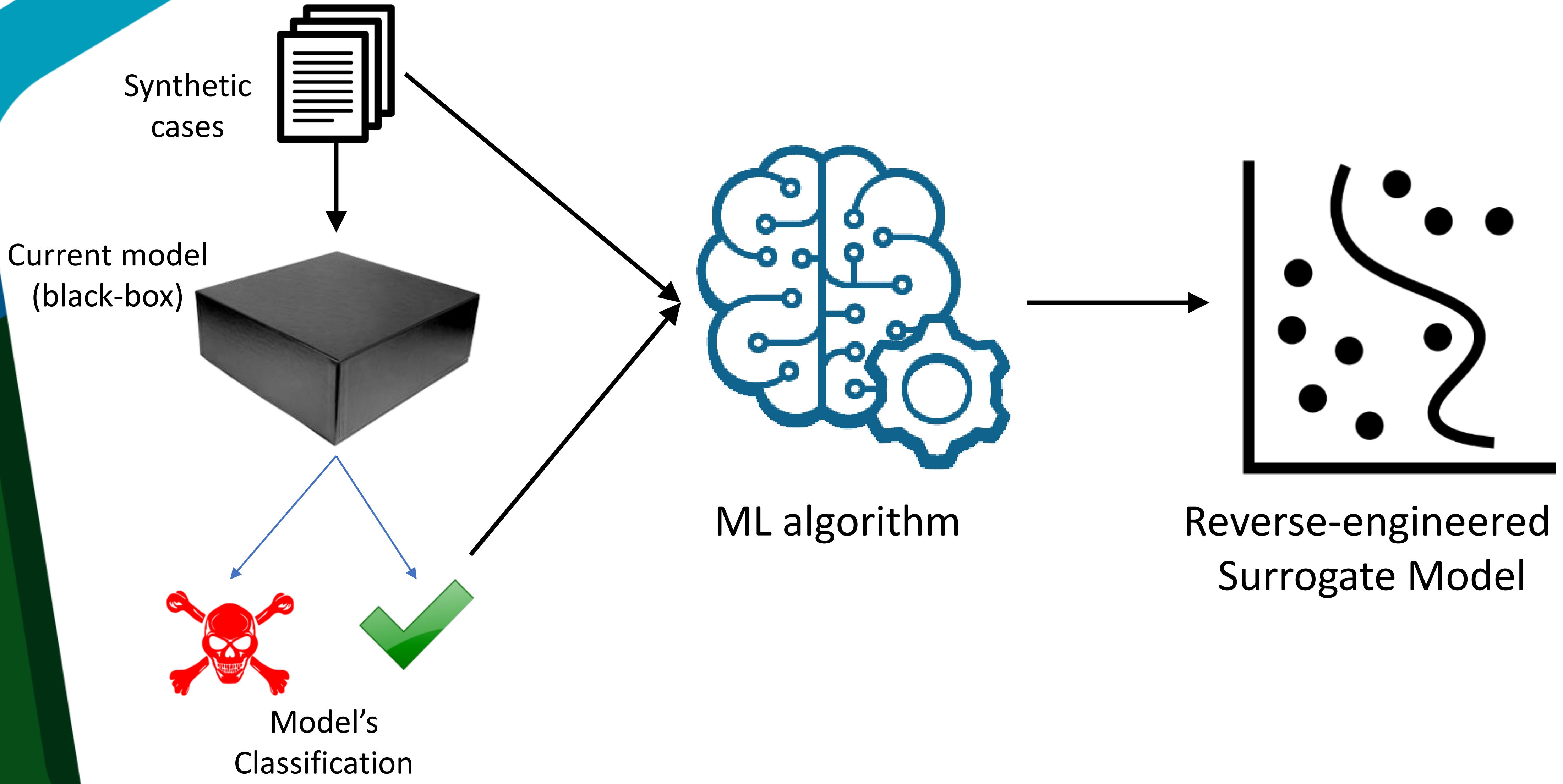
Malware classification process


Inspected File
VirtualBox + Cuckoo Sandbox
JSON Report
Parser
One-Hot Encoded API Call Sequence
Classifier
Malicious or Benign Classification
Reporting

Research outline

Detection method	Goal	Method
• **Dynamic analysis** classifier, using Windows **API (including OS) calls** as features	• **Modify malicious code** to be **classified** as **benign**; **without sacrificing** its **malicious functionality** • **Assumptions** • No knowledge about the classifier • Query the classifier as a black-box • Knows about the representation of the API calls	• Two-phase **black-box attack** that **exploits** the **transferability property** • training a surrogate model, using the original model as a black-box • creating an adversarial example against the surrogate model

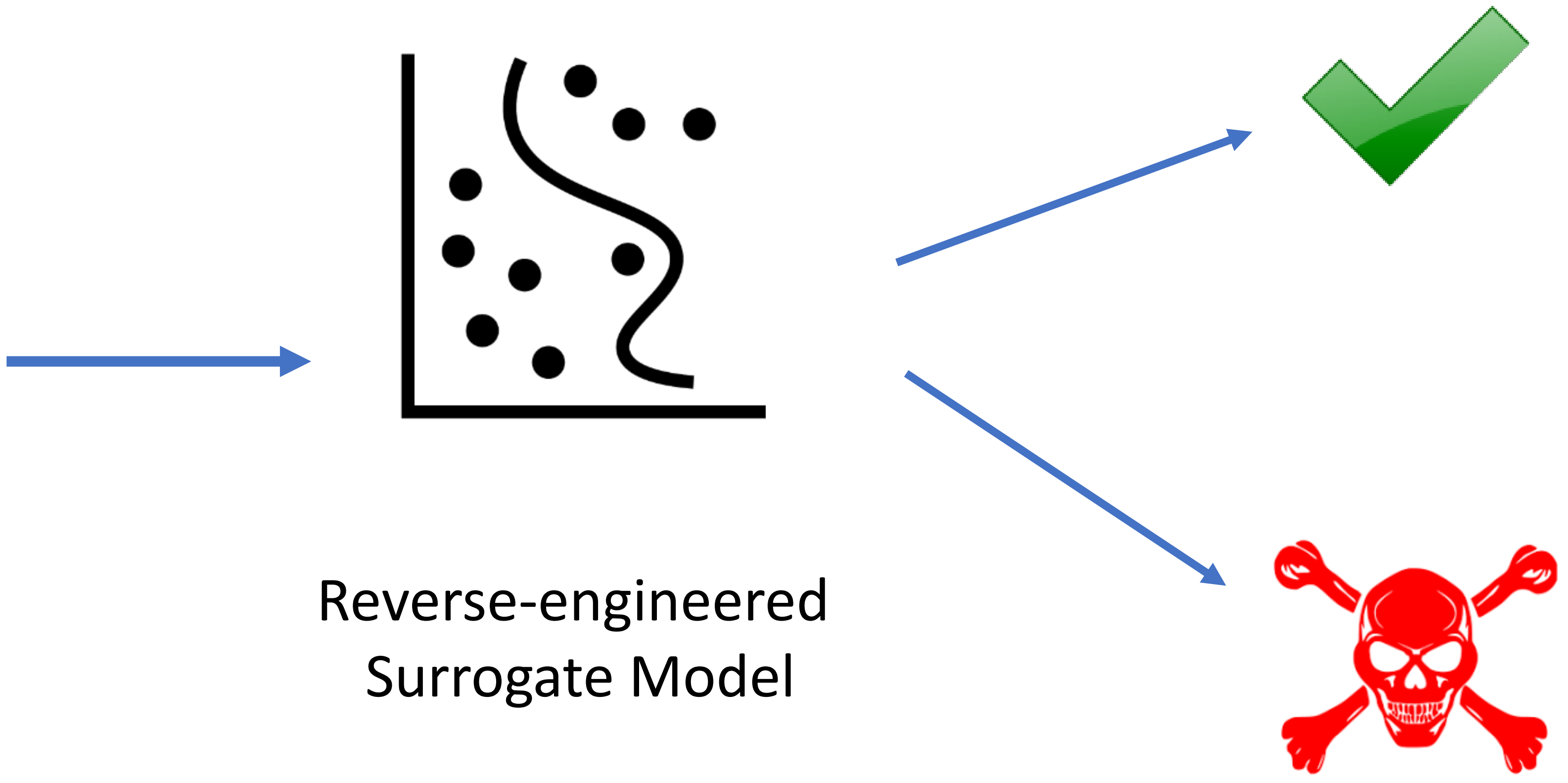
Phase I: Generating the Surrogate model



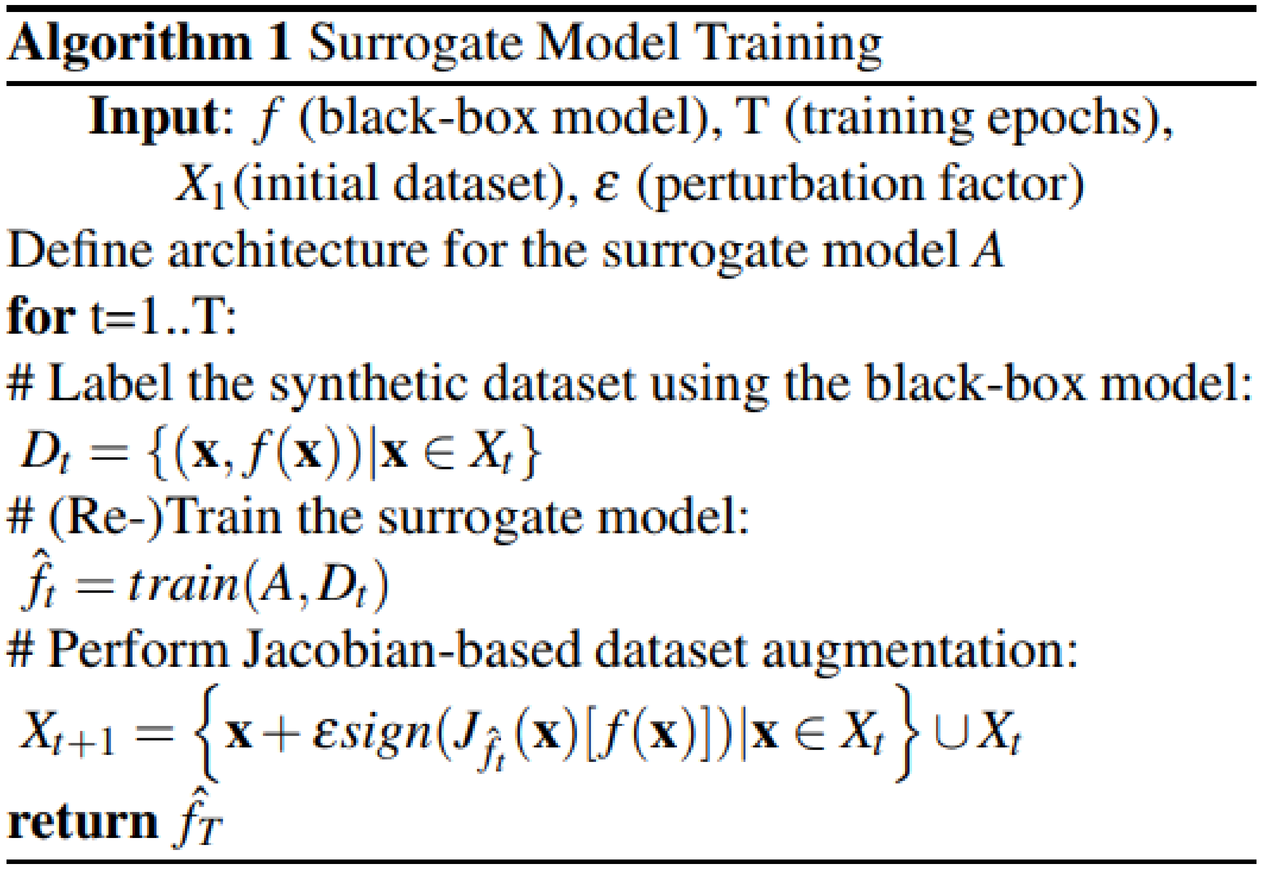
Credit: https://www.udacity.com/di-tech and www.iconsdb.com

Phase II: Creating an adversarial example against the surrogate model

```
creat()
open()
close()
read()
write()
lseek()
dup()
link()
unlink()
stat()
fstat()
access()
chmod()
```



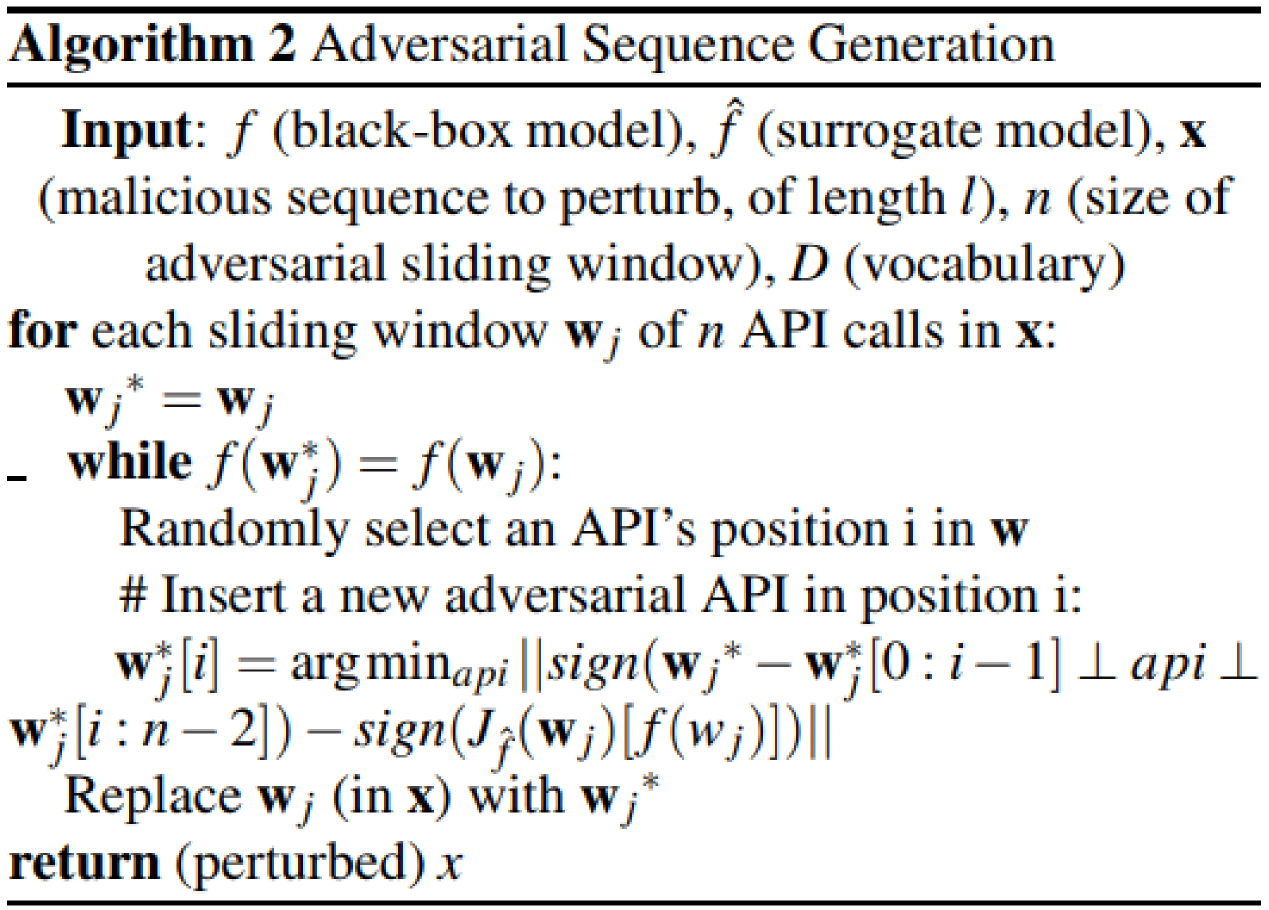
- Start with a small sample of files (in our experiments 70)
- Repeatedly
  - generate synthetic samples by a Jacobian-based heuristic; used for approximating the black-box model's decision boundaries
  - synthetic samples are labeled by the black-box model and used for re-training the surrogate model
- *in our experiment we generated in total 2,240 samples (compared to 360,000 used for training the black-box model!)*



Algorithm 1 Surrogate Model Training

```
    Input: $f$ (black-box model), T (training epochs),
        $X_1$ (initial dataset), $\varepsilon$ (perturbation factor)
Define architecture for the surrogate model $A$
for t=1..T:
# Label the synthetic dataset using the black-box model:
$D_t = \{(\mathbf{x}, f(\mathbf{x})) | \mathbf{x} \in X_t\}$
# (Re-)Train the surrogate model:
$\hat{f}_t = train(A, D_t)$
# Perform Jacobian-based dataset augmentation:
$X_{t+1} = \left\{\mathbf{x} + \varepsilon sign(J_{\hat{f}_t}(\mathbf{x})[f(\mathbf{x})]) | \mathbf{x} \in X_t\right\} \cup X_t$
return $\hat{f}_T$
```

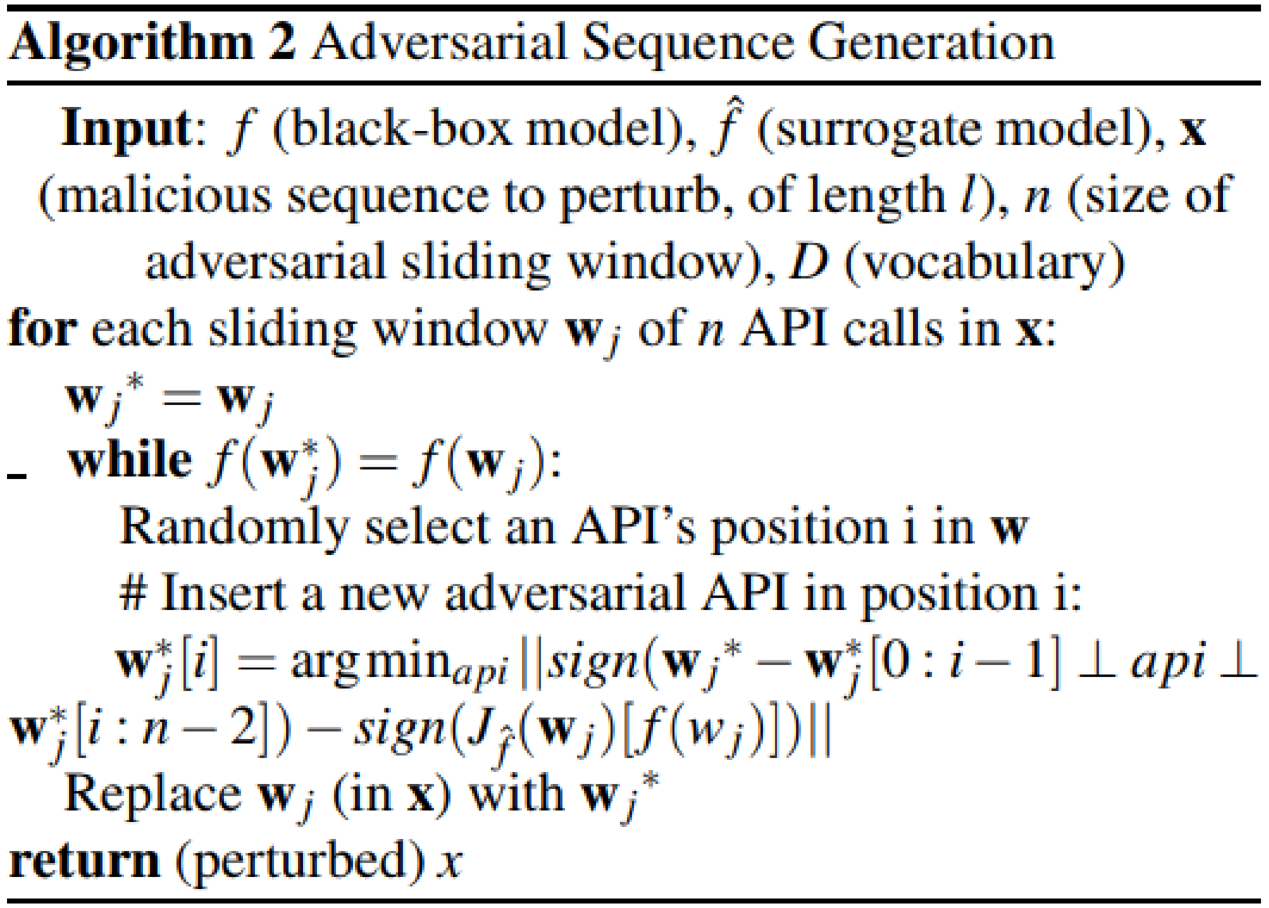
- Analyze long sequence of API calls (*l*)
- Use sliding window on *n* API calls
- If at least one window is classified as malicious, the application is labeled as malicious



Algorithm 2 Adversarial Sequence Generation

```
Input: $f$ (black-box model), $\hat{f}$ (surrogate model), $\mathbf{x}$ (malicious sequence to perturb, of length $l$), $n$ (size of adversarial sliding window), $D$ (vocabulary)
for each sliding window $\mathbf{w}_j$ of $n$ API calls in $\mathbf{x}$:
    $\mathbf{w}_j{}^* = \mathbf{w}_j$
    while $f(\mathbf{w}_j^*) = f(\mathbf{w}_j)$:
        Randomly select an API's position i in $\mathbf{w}$
        # Insert a new adversarial API in position i:
        $\mathbf{w}_j^*[i] = \arg\min_{api} ||sign(\mathbf{w}_j{}^* - \mathbf{w}_j^*[0:i-1] \perp api \perp \mathbf{w}_j^*[i:n-2]) - sign(J_{\hat{f}}(\mathbf{w}_j)[f(w_j)])||$
    Replace $\mathbf{w}_j$ (in $\mathbf{x}$) with $\mathbf{w}_j{}^*$
return (perturbed) $x$
```

- For each windows, randomly select an API call and replace it with another API call (that will not impact the normal operation of the application – *no-op attack*)
- Select the new API call that goes in the direction of the Jacobian; thus allowing to minimize the number of changes required



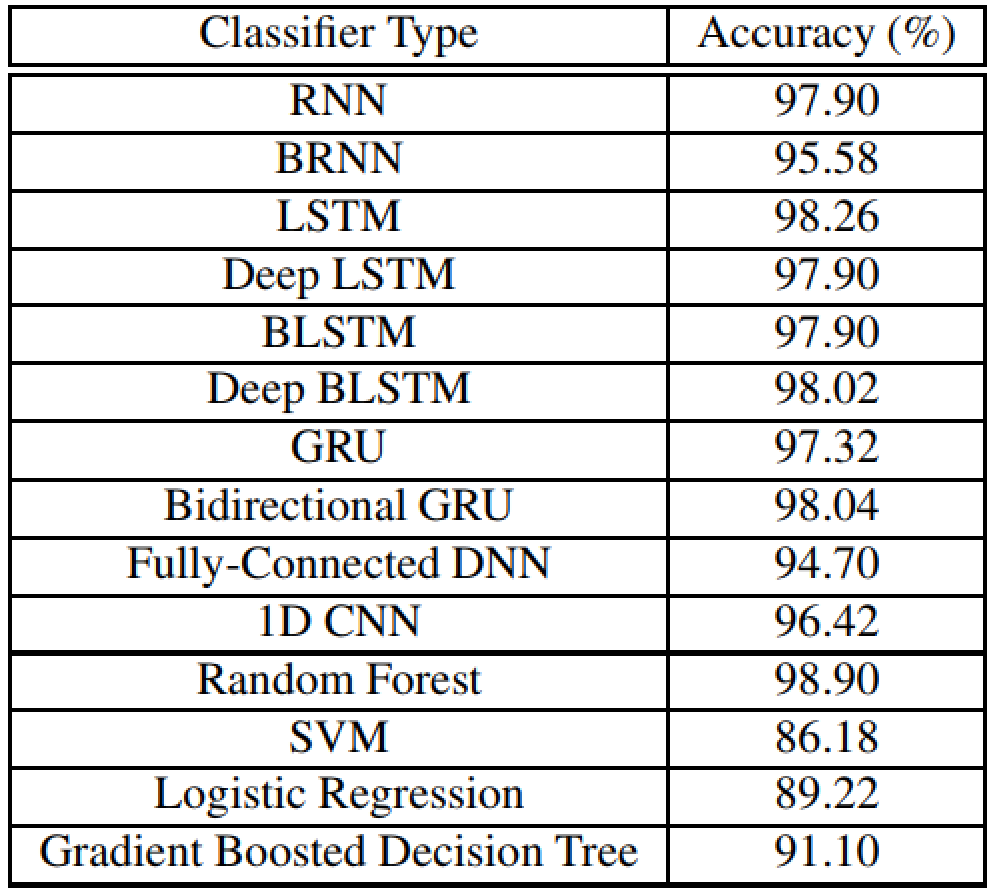
Algorithm 2 Adversarial Sequence Generation

```
Input: $f$ (black-box model), $\hat{f}$ (surrogate model), $\mathbf{x}$ (malicious sequence to perturb, of length $l$), $n$ (size of adversarial sliding window), $D$ (vocabulary)
for each sliding window $\mathbf{w}_j$ of $n$ API calls in $\mathbf{x}$:
    $\mathbf{w}_j{}^* = \mathbf{w}_j$
    while $f(\mathbf{w}_j^*) = f(\mathbf{w}_j)$:
        Randomly select an API's position i in $\mathbf{w}$
        # Insert a new adversarial API in position i:
        $\mathbf{w}_j^*[i] = \arg\min_{api} ||sign(\mathbf{w}_j{}^* - \mathbf{w}_j^*[0:i-1] \perp api \perp \mathbf{w}_j^*[i:n-2]) - sign(J_{\hat{f}}(\mathbf{w}_j)[f(w_j)])||$
    Replace $\mathbf{w}_j$ (in $\mathbf{x}$) with $\mathbf{w}_j{}^*$
return (perturbed) $x$
```

Evaluation

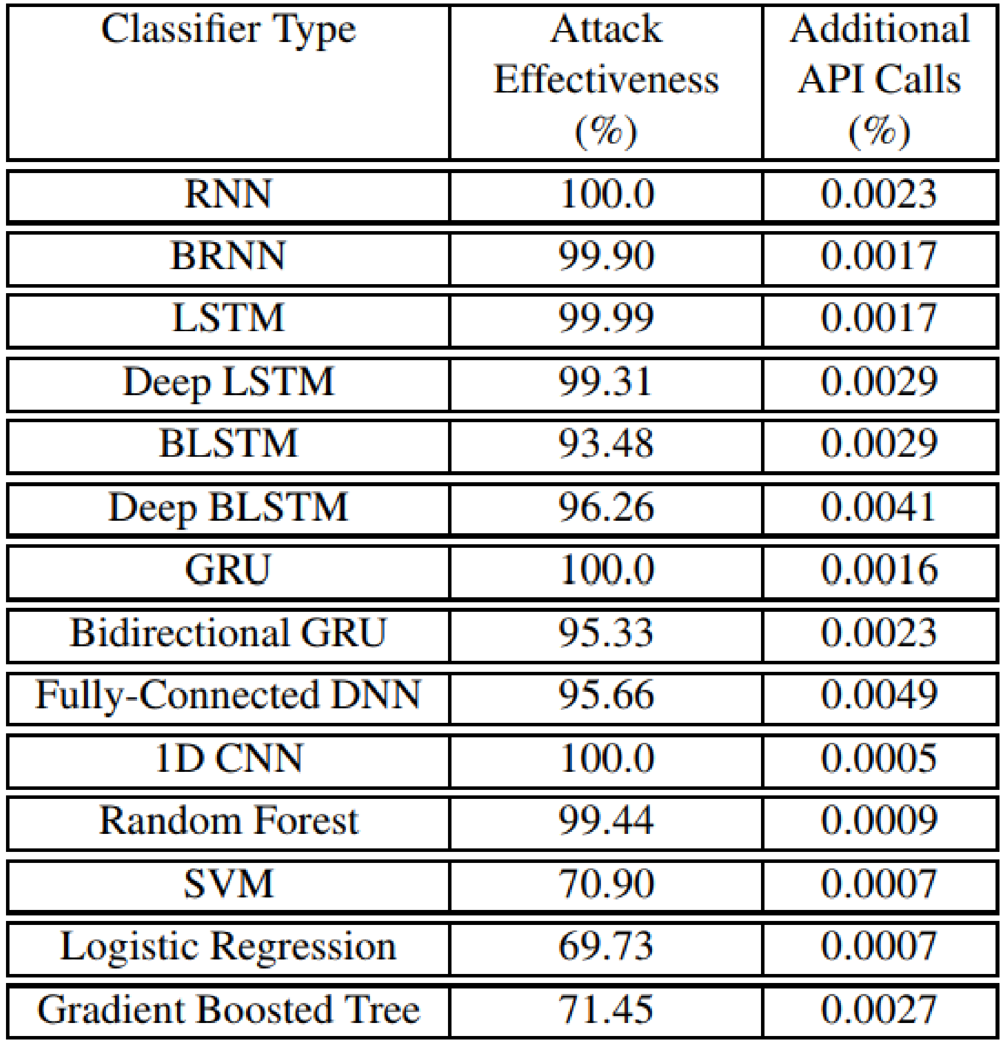
- **Dataset**:
  - Training set containing 360K Cuckoo Sandbox reports (balanced)
  - Test set containing 36K Cuckoo Sandbox reports (balanced)
- **Surrogate model**
  - Gated Recurrent Unit (GRU), 64 units
  - 70 randomly selected samples from the original test set
  - They were excluded from the test set
  - 6 surrogate training epochs (T=6) – total of 2,240 samples
- For **window size**, we used the first 140 API calls (Pascanu *et al*., 2015)
  - Also tried 100, 120 and 1,000
  - Average length of API calls sequences *l* = 100,000
  - 100 Training epochs, early stopping
- **API calls were one-hot encoded**
  - For generality

Classifiers performance



Classifier Type	Accuracy (%)
RNN	97.90
BRNN	95.58
LSTM	98.26
Deep LSTM	97.90
BLSTM	97.90
Deep BLSTM	98.02
GRU	97.32
Bidirectional GRU	98.04
Fully-Connected DNN	94.70
1D CNN	96.42
Random Forest	98.90
SVM	86.18
Logistic Regression	89.22
Gradient Boosted Decision Tree	91.10

Attack effectiveness



Classifier Type	Attack Effectiveness (%)	Additional API Calls (%)
RNN	100.0	0.0023
BRNN	99.90	0.0017
LSTM	99.99	0.0017
Deep LSTM	99.31	0.0029
BLSTM	93.48	0.0029
Deep BLSTM	96.26	0.0041
GRU	100.0	0.0016
Bidirectional GRU	95.33	0.0023
Fully-Connected DNN	95.66	0.0049
1D CNN	100.0	0.0005
Random Forest	99.44	0.0009
SVM	70.90	0.0007
Logistic Regression	69.73	0.0007
Gradient Boosted Tree	71.45	0.0027

GADGET: Generative Api aDversarial Generic Example by Transferability framework

- A framework demonstrating an attacker's ability to create and end-to-end attack using the proposed method
- *no-op* attack – adding API's which have no effect on the code's functionality; with valid parameters
- Add the adversarial modifications as an input
- The injected code would trace the APIs called so far and each injected API would also call the adversarial added API, if needed

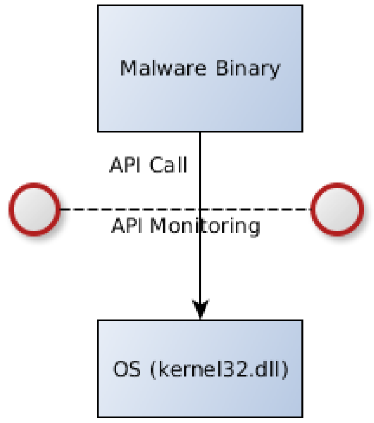
Advantages:

Requires no access to the malware source code

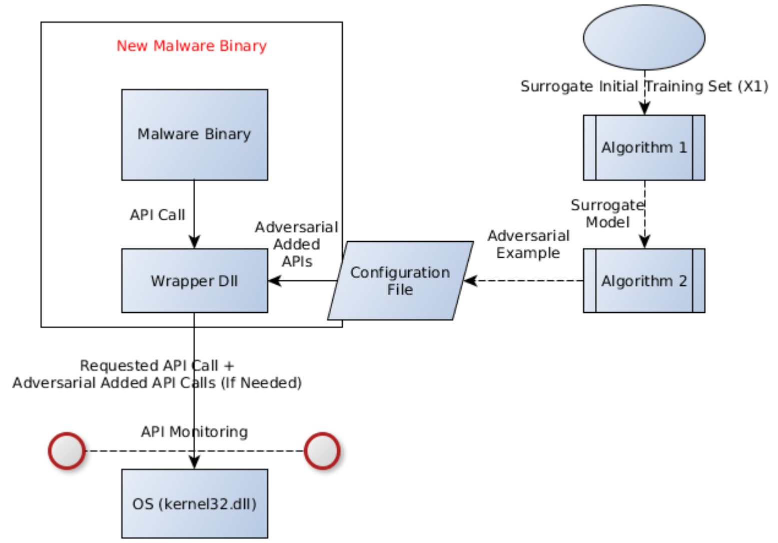
Generic – fit every code

Disadvantages:

API call type-specific



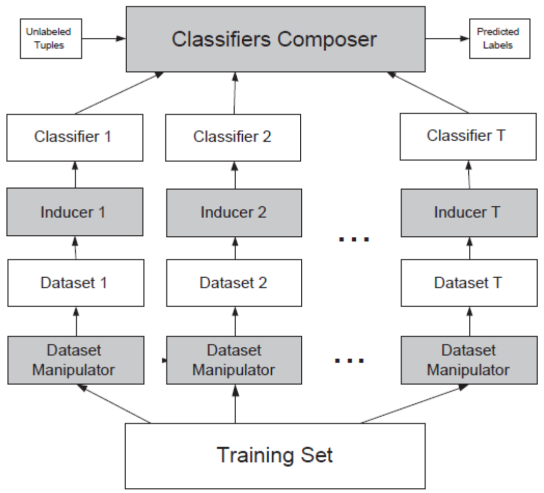
Malware Binary, without GADGET



Malware Binary, with GADGET

Possible defense mechanisms

- Statistics of API calls [Grosse *et al*., 2016]
- Use an additional adversarial example detection model [Metzen *et al*., 2017]
- Adding more input features (Wang *et al*., 2016)
- Adding adversarial examples to the training set; e.g., using GANs [Szegedy *et al*., 2014]
- Use a multiple classifier system; each classifier is trained on different training set



Main contributions

- Novel **end-to-end** black-box attack, preserving the malware functionality after perturbation
- **Generic attack** - effective against conventional RNN, LSTM, GRU, bidirectional and deep variants, fully-connected DNN, 1D- CNN, SVM, logistic regression, random forest, GBDT
- Access to the malware **source code is not required** (using GADGET, end-to-end framework to modify the malware)
- Bypasses **multi-feature classifiers** using a combined attack to fit real world scenarios
  - e.g., hybrid classifiers based on printable strings and API sequences
- First to **evaluate transferability** between RNN variants, feed-forward networks and traditional machine learning classifiers

Conclusions

- The main focus of the AI community has been on making AI models accurate; in the meantime, these models were left vulnerable, representing a concrete threat to AI safety
- Therefore, practitioners should make the models resilient to adversarial attacks (throughout the supply-chain) before embedding AI technologies in the physical world
- This is extremely important for self-learning systems

Thank you