SWAT: Seamless Web Authentication Technology

Florentin Rochet
UCLouvain Crypto Group
florentin.rochet@uclouvain.be

Kyriakos Efthymiadis
AI Lab
Vrije Universiteit Brussel

François Koeune
UCLouvain Crypto Group
francois.koeune@uclouvain.be

Olivier Pereira
UCLouvain Crypto Group

ABSTRACT

We present a seamless challenge-response authentication protocol which leverages on the variations of html5 canvas rendering made by the software and hardware stacks. After a training phase that leads to feature extraction with deep learning techniques, a server becomes able to authenticate a user based on fresh canvasses, hence avoiding replay attacks. The whole authentication process is natively supported by any mainstream browser, stateless on client side and can be transparent to the user. We argue that those features facilitate deployment and composition with other authentication mechanisms without lowering the user experience. We present the threat model against which our protocol is expected to live and discuss its security. We also present a prototype implementation of our protocol and report on a real-word experimentation that we ran in order to analyze its efficiency and effectiveness.

KEYWORDS

convolutional neural networks, authentication, html5 canvas, 2FA, usable security

ACM Reference Format:
Florentin Rochet, Kyriakos Efthymiadis, François Koeune, and Olivier Pereira. 2019. SWAT: Seamless Web Authentication Technology. In *Proceedings of the 2019 World Wide Web Conference (WWW'19), May 13–17, 2019, San Francisco, CA, USA.* ACM, New York, NY, USA, Article 4, 11 pages. https://doi.org/10.1145/3308558.3313637

1 INTRODUCTION

Although paramount for security, user authentication is often a hindrance for users: typing in a password, connecting a security token or interacting with a dedicated device is seen as a frustrating and time-consuming step. As a result, users more often than not are tempted to use time-saving shortcuts – bad or identical passwords, access card left next to device… – damaging for security, or to give up the transaction if the "reward" is not deemed to be worth the effort, resulting in losses for the provider. In this sense, seamless authentication protocols represent an appealing target.

Recently, browser fingerprinting [4], and in particular canvas fingerprinting [19] have been introduced as a means of identification. Put simply, the idea is that browser-specific features such as the supported fonts, software version or, for canvas fingerprinting, rendering of vector graphics can provide sufficient information to allow distinguishing a device from another, and hence to recognize a user coming back after a previous visit. Yet, to the best of our knowledge, this technique has been used only for identification – in the sense of recognizing a user – and not for authentication – in the sense of gaining confidence that a user is indeed who he pretends to be.

This paper represents a first attempt in this direction. Using canvas fingerprinting and deep learning techniques, we propose a challenge-response protocol taking place between a server and a standard web browser and operating in way that is transparent for the user. Intuitively, in an initial phase, the server asks the browser to perform several drawings and, through the use of deep learning, "learns" specifics about how the browser draws. As part of a later authentication process, the browser receives from the server a challenge describing a new drawing to perform. The server then passes the new drawing through the, now trained, deep learning model and if it recognizes the same specifics in the resulting image, authentication is deemed successful. The challenge-response process of SWAT aims at offering protection against stolen credentials, or offline replay attacks by malicious servers.

Our contributions are the design, the implementation and the evaluation of SWAT, a new protocol for user-friendly authentication based on canvas extraction of hardware/software dependencies. More specifically, we offer:

- An open-source implementation of SWAT for the Django Web framework. Our implementation covers (i) the data collection needed for the machine learning phases, (ii) the learning phase itself, implemented as a cron job that verifies every minute if new users registered and (iii) the authentication phase if a learned model is available for a given user.
- Experiments and evaluation of the efficacy of SWAT given real-world collection of user data.
- A discussion about real-world applications of SWAT, as well as its expected security benefits. We emphasize SWAT's security in a threat model that shows an interesting complementary with standard authentication systems, supporting the use of SWAT as a second authentication factor.

Our protocol takes advantage of standardized web technologies. In consequence, SWAT is natively supported by Web Browsers and is designed to be fast and easy to deploy.

Roadmap. Section 2 provides the necessary background for canvas fingerprinting and deep neural networks. Then, we describe the

ACM ISBN 978-1-4503-6674-8/19/05.
https://doi.org/10.1145/3308558.3313637

SWAT protocol: how the client and server side conjointly perform the authentication as well as the threat model supported by our protocol. Section 4 covers an experimental analysis of our protocol, with real-world collection of data and analysis of the accuracy of the authentication. Section 5 describes various usage scenarios. Finally, Sections 6-8 discuss related work and future work before concluding.

2 BACKGROUND

2.1 Canvas fingerprinting

Mowery and Shacham [19] designed a new fingerprinting method based on browser font and WebGL rendering. This method exploits the surprising output variation of the HTML5 `<canvas>`. The `<canvas>` element provides, within a web page, an area in which the programmer can draw using Javascript. The variation depends on the operating system running the browser, the type of CPU or GPU, the graphic driver, the browser brand and its version.

The fingerprinting method works as follows: some Javascript code recovers the context of a canvas element within the webpage and draws a text and a text's scene. The goal is to generate a drawing impacted by as much software and hardware components as possible. The programmer, using common high level API functions, produces a drawing and then converts it to a base64 representation of the image. Then, a collision-resistant hash function receives this representation as an input and outputs a different value (the fingerprint) for any small change within pixel position or color of the image.

This method has been extensively used as a stateless tracker [1, 5], because the same user should produce the same hash output when drawing the same text and text's scene twice. In this paper, we try to take advantage of canvas's variation to the benefit of web users.

2.2 Deep Neural Networks

A natural candidate for extracting useful information from canvas drawings in an automated fashion is deep learning. Deep learning is a subfield of machine learning research which has demonstrated huge success in computer vision tasks like classification, localisation and segmentation [12, 13, 15, 27, 28].

Typically in machine learning, algorithms receive input features and produce, hopefully, useful outputs. The input features are usually manually defined and depend on the application domain. Consider for example an image classification task in which the algorithm needs to decide what object is displayed in an image. The input in this case would be defined in terms of a set of features defined on the pixels of the input image. It is easy to realize that defining manual features in the pixel space is extremely difficult. Representation learning can help alleviate this problem by providing the means to build models that can extract useful information from data.

Deep learning is a type of learning representation which follows a structured approach to extracting information from input data. It accomplishes this by using several layers to build representations of increasing complexity with initial layers dealing with primitive types, while the next layers deal with more complex objects composed of primitive types. For more in depth information regarding deep learning we refer the reader to Goodfellow *et al.* [10].

2.2.1 *Fully Connected.*

Figure 1 shows a typical fully connected network. Its layers are comprised of scalar output values called *neurons* which are arranged in a hierarchical fashion. Information flows from the input all the way to the output with in-between layer outputs being the input to the following layer. Information does not flow back hence this network structure is also called a *feed-forward network*. The output of an individual neuron is called its activation which is typically non-linear (deep networks of linear activations are equivalent to single layer networks), hence it is referred to as the *non-linearity*. The activations of the last layer (output) is the prediction of the network given the first layer (input); all other layers are not directly observed and are conveniently called *hidden layers*.

Let $X = (x_1, x_2, x_3, \ldots, x_n)$ be a vector of inputs and $w = (w_1, w_2, w_3, \ldots, w_n)$ a vector of weights and b a bias. Then the output, or activation, of a given neuron is calculated by

$$a = \sigma(w \cdot x + b), \tag{1}$$

in which σ is the non-linearity. Currently the most used non-linearity is the rectified linear unit, or ReLU [21]. The ReLU function is defined as $\sigma(x) = \max(0, x)$ and it has been shown to result in strong gradients that allows much faster training for deep networks.

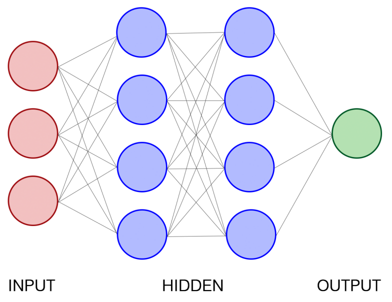


Figure 1: Fully Connected Neural Network.

What we want to find is the set of weights and biases that bring the outputs of the network closer to the desired output. Specifically, given network predictions $\hat{y} = (\hat{y}_1, \hat{y}_2, \hat{y}_3, \ldots, \hat{y}_n)$ and targets $y = (y_1, y_2, y_3, \ldots, y_n)$, we want the network outputs to approximate the targets as closely as possible for all inputs $X = (x_1, x_2, x_3, \ldots, x_n)$. This can be formulated as an optimization problem where we try to minimize a loss function. In this work we are dealing with a binary classification problem hence we use the cross-entropy loss to train our model over all training examples M,

$$Loss = -\frac{1}{M} \sum_{i}^{M} \left[y_i \log \hat{y}_i + (1 - y_i) \log(1 - \hat{y}_i) \right] \tag{2}$$

In order to find the model parameters that minimise the loss function we use gradient descent. By taking the gradient of the loss function w.r.t. the model parameters we can find how these parameters need to be changed to reduce the loss value. The parameters are initialised at random and are then iteratively updated by descending along the gradient surface until a suitable minimum value is reached. Of course computing the exact gradient for the loss function would have us iterate through the entire dataset which is prohibitively expensive. Instead what is typically used is mini-batch gradient descend in which a subset of the training data is used to approximate the gradient at each step.

2.2.2 Convolutional Neural Networks. Convolutional neural networks [8, 18], CNNs, is the state-of-the-art technique when dealing with problems of computer vision in machine learning. Typical, fully connected networks, in which every neuron in layer l is connected to every neuron in layer $l+1$ ignore a very important aspect when dealing with images as input; spatial information. Every pixel in a fully connected network is treated independently when in most cases, pixels spatially located close to each other is important information that needs to be captured. CNNs provide the means to exploit the local structure in the input by learning local features. In deep architectures, earlier convolutional layers will learn to extract spacial features such as lines, edges and so on, while later layers will learn to combine those to form more distinct features, such as a face or a dog.

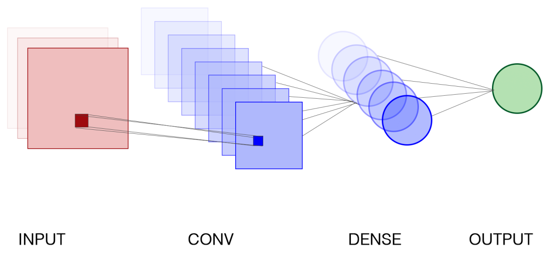


Figure 2: Convolutional Neural Network.

Figure 2 shows a schematic representation of a CNN. Typically a CNN is built by a series of convolutional layers, non-linearities, pooling layers and finally the output. The input to the CNNs presented in this work is an image. At each layer the input is convolved with a set of weights called *filters* to produce *feature maps*. Different feature maps i.e. layer outputs, are convolved with different sets of filters, but filters are shared within the same feature map. It is those sparse interactions and parameter sharing that greatly reduce the number of parameters of the model, its computational and memory complexity.

Equation 3 show the operation that defines the l-th feature map

$$a^l = \sigma\Big(\sum_m w_m^l * x + b^l\Big), \tag{3}$$

where we sum over all features maps, w_m^l are the filter weights to be learned, b is the bias, $*$ represents the cross-correlation operation (in deep learning known as "convolution"), and σ represents the element-wise activation function which as discussed previously is typically a ReLU. Popular kernel choices in most cases, and dependent on the input size, are $3 \times 3, 5 \times 5, 7 \times 7$. A pooling layer usually follows which aggregates the values of neighboring neurons based on their maximum, or average value. This results in the model being invariant to small translations in the input and is an important aspect if we only care about the presence of a feature and not its placement. Typically pooling layers have a stride of 2 i.e., they aggregate values that are spaced 2 neurons apart. This results in a reduction of neurons allowing the following convolutional layer to have a larger field of view of the input. For more in depth information regarding convolutional networks and their applications we refer the reader to [10].

3 SWAT

3.1 Overview

Authentication is usually based on an agreed token that is recognized by an identity provider and that a user uses to prove this identity. This token can take many forms with many different extraction methods. These are commonly divided in the following well-known classification: something we know, something we have and something we are.

In our case, the token can be seen as "the way the user draws in canvas" and by the user, we mean his/her full stack of hardware and software. Our method to extract this token and to send it toward the identity provider shows a unique advantage: it requires no interaction with the user and all recent browsers are compatible without any additive software to plug-in. This technique falls into the "something we have" category, however offering a better usability than other methods classified into the same category.

The SWAT authentication protocol is a challenge-response protocol that takes advantage of canvas's high variation and deep learning's feature extraction to provide a user-friendly layer of authentication. We can abstract SWAT as three independent phases: P1: data collection, P2: learning phase and P3: authentication phase. Phases P1 and P2 must be completed once for each stack of hardware and software owned by a user. When P1 and P2 are complete, the authentication phase can be performed as often as needed.

The data collection phase can be performed in a stealthy way to the user. While we advocate for transparency, scenarios may exist where a stealth collection of data is preferable. Such choice would be left to the identity provider, but could also be integrated as part of user preferences.

In our proof of concept implementation, the browser generates 2000 35x280x4 images based on random texts sent by the server (as a matter of fact, writing in canvas appears to be one of the most browser-sensitive features we observed). We extract those images from a <canvas> element, and convert them to a base64 representation which we send to the server.

Once the data collection (P1) is finished, we start extracting features in the learning phase (P2). We use a CNN to extract the user's features and build a binary classification model for each user. This is in contrast to what we usually see in the literature where one end-to-end model is trained with multiple classes. The reason we have decided to follow the path of per-user model is rooted in a more efficient training procedure for our authentication system. If we employed an end-to-end model trained on all users, then every time a new user would register to our system, we would need to re-train the entire network; a process which takes time and resources. By having a binary classification process where we train a model per user, we do away with that problem. Our current CNNs are simple enough to spend less than 3 minutes of learning with a decent accuracy and almost no over-fitting. We have designed our CNN this way to focus on usable research and show that this authentication method could be deployed, even at a large scale for systems handling a large pool of users.

Finally, in the authentication phase (P3), the browser and the server play a (stealthy) challenge-response protocol, where the server asks to draw several texts and text's scenes. The server then uses the binary classification built during P2 to assess whether the browser's answer matches its user's model. To easily verify the validity of the client's response and prevent a replay attack, the server produces locally the same drawing as the one he asked from the browser and compares it to the response received from the browser using the method of least squares. Even though the two images will not be identical – this is the very essence of SWAT – they should be close enough for the least square difference to be small when based on the same challenge.

We provide in this paper a full stack implementation of the protocol outlined above and we describe, in the section below, each piece in details. We do not see SWAT as a replacement for existing strong authentication methods, as it does not guarantee as much security as a strong password for instance. However, we believe that SWAT is an interesting candidate to be part of a layered multi-factors authentication system, such as password+SWAT or other methods, mostly depending on the level of security needed and the threat model.

3.2 Client side

The client side of the SWAT protocol is mostly responsible for 1) drawing the random canvas asked by the server and 2) extracting a representation of the canvas as a png image. The Javascript code then sends the image representation to the server. As thousands of images are generated and sent during the collection phase, it might take up to a few minutes to complete. A "stop and resume later" strategy for the collection phase is possible, but the user needs to authenticate with another method to resume.

```javascript
function generateRandomCanvas(txt) {
  var canvas =  document.getElementById("canvas");
  canvas.width = 280, canvas.height = 35;
  var ctx = canvas.getContext("2d");
  ctx.clearRect(0,0, canvas.width, canvas.height);
  //drawing the new canvas
  ctx.font= "18px 'Arial'";
  ctx.textBaseline = "alphabetic";
  ctx.fillStyle = "#069";
  ctx.fillText(txt, 2, 15, 280);
  ctx.fillStyle = "rgba(102, 204, 0, 0.7)";
  ctx.fillText(txt, 4, 19, 280);
  ctx.fillStyle = "#069";
  ctx.fillText(txt, 2, 23, 280);
  ctx.fillStyle = "rgba(102, 204, 0, 0.7)";
  ctx.fillText(txt, 4, 27, 280);
  ctx.fillStyle = "#069";
  ctx.fillText(txt, 2, 31, 280);
  ctx.fillStyle = "rgba(102, 204, 0, 0.7)";
  ctx.fillText(txt, 4, 35, 280);
}
```

Listing 1: javascript code to generate a canvas content

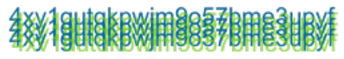


Figure 3: Example of a canvas drawn by SWAT

During the authentication phase, the client receives a few texts to draw from the server (E.g. Figure 3). This process takes less than a second, does not require any interaction with the user and can be designed to be invisible.

3.3 Server side

The prototype implementation we developed [25] is a Django plugin using the Python deep learning *Keras* API [14] as the front-end to the server-side programmer and *Tensorflow* as the back-end. The collection phase stores every *base64* encoded *.png* image into a database for each user. Once we have enough different users, the system starts to learn by pooling each user's generated images and a set of random images from other users. For each user, we have 2000 images labelled as "yes" (all collected images) and 2551 random images from the database labelled as "no". We choose those numbers such that, when the system splits the set of images into a training set and a validation set, we obtain a training set size as a multiple of the batch size for efficiency reasons.

The system trains and saves one model per user as a binary classification model. During the authentication phase, we load the model associated to the claimed user's identity and we feed the model with the few fresh images that we asked to draw. For each input image, the model outputs a prediction value. In theory, if the claimed user's identity is right, the predicted value should be high. The closer the prediction is to 1, the more confident we are about the image belonging to the right user.

A user identity may have multiple learned models, namely one for each of his stacks of hardware/software (e.g. chrome browser and firefox on his laptop). The user should succeed the authentication for one of the learned model but not for the others.

3.4 Threat model and security

We now discuss the type of adversary against whom SWAT is resistant as well as how the protocol could be combined with other forms of authentication to protect against several threats without giving up on usability.

In contrast to security notions provided in cryptography, the non-forgeability of valid canvas for a given user cannot be linked to a computationally hard problem. The forgeability of SWAT can be reduced to the following adversarial game: given a set of valid canvasses for an identity and given a challenge, draw a canvas that obtains a high prediction value with non-negligible probability. If the adversary wins this game, then we have an active attack against SWAT. Assessing the resistance of SWAT against such an active adversary is an open research problem, which involves analyzing how deep learning (and other) strategies could allow him to reconstruct the model based on partial information and with a non-negligible success probability. As a preliminary step, in this paper we consider only the case of passive attackers, that is, attackers who follow the protocol but do not actively try to forge a canvas based on a more sophisticated strategy.

The SWAT protocol aims to protect against the following problems:

- **Passive eavesdropping and replay attacks**: since SWAT is a challenge-response protocol, a consumed canvas cannot be replayed to gain access to the authenticated channel. Therefore, the SWAT protocol may be interesting to use in addition to other authentication methods that do not protect against those problems.
- **Stolen credentials**: The SWAT protocol is a stateless protocol, hence the information needed to authenticate cannot be collected on a user device.

Under the assumption that an indistinguishable forgery cannot be performed, our canvas challenge-response approach mitigates the following issues:

- **Database leakage**: If the database of a service gets compromised, it should not give the opportunity to an attacker to forge a valid canvas and obtain a valid credential.
- **Website attacker**: If a website collects drawings against the will of a user, it should not help to forge a valid canvas, except through an on-line man-in-the-middle attack. As a partial mitigation, we observe that Firefox recently added a user approval on canvas extraction [7] based on a Tor Browser's functionality which would reduce the likelihood of such risk.

Some threats remain out of scope for this protocol:

- **Active man-in-the-middle attack**: such as a website attacker that tricks a user into completing a challenge for another service. Due to the design of SWAT, the tricked user would not notice any completed challenge toward another service.
- **Hardware/software cloning**: given the software stack of a target, and given the information related to the exact hardware the target owns, an adversary would be able to build a machine that would successfully authenticate him.

This protocol is not designed for high security and we would discourage a standalone use to protect sensible information. However, this protocol closes the security gap that some other authentication methods have, such as weak passwords. More generally, when a protocol relies on an attached secret to perform the authentication, it is always weak against the leakage of such a secret. SWAT may prevent the leakage of a password, a cryptographic key or the theft of a security token from providing the thief with a direct access to the secure environment. This protocol may offer the time needed to the user to react and recover new strong credentials.

4 EXPERIMENTAL ANALYSIS

4.1 Methodology

Our main objective is to obtain a seamless, user interaction-free layer of authentication which user intensive applications could use to enhance their account security system. For this purpose, the main constraint we impose to the evaluation of our prototype is to have a relatively fast training phase to comply with realistic utilization of this protocol. By realistic utilization, we mean realistic training cost and sufficient efficacy.

We started by developing a framework for the multi purposes of data collection and evaluation, as well as for protocol demonstration. We chose CNN architectures in such a way that the learning phase takes less time than the collection phase (less than 1 minute). We do not argue in this paper for an optimal architecture suited for our protocol, since we believe that the optimum is bounded by the resource availability of the identity provider: the more resources the identity provider owns, the more complex CNN architectures they could employ. We made the choice to experiment with two different architectures to demonstrate the trade-off between result efficacy and cost of the training phase.

We opened the data collection process within the main author's social network, and we explain in Section 4.2 the steps taken to avoid any pollution (i.e., injection of wrong data) of the database. We evaluate our architectures' accuracy by splitting the set of available canvases for a given user in a training set and a validation set. During the evaluation, one of our goals was to study the impact of having common pieces of software on the prediction value. For example, what is the chance for user A running Mac OS X to score a high prediction on behalf of user B also running Mac OS X? To answer those questions, we collected meta-information about participating users and ran the authentication phase for each user over every other user in our dataset.

We finally investigate different solutions to accept or reject authentication attempts from the output of the classifier.

4.2 Data Gathering

We built a web interface [26], to gather real user data over a few weeks. In total, 118 collections have been done and among them, 109 fully completed the collection phase. We discarded the uncompleted involvements for this analysis, leaving us 109 remaining datasets. Alongside the 2000 images collected for each participant, we collected some metadata such as the operating system family and the browser family.

The data gathering has been made public on social networks, hence we had to design the gathering with security issues in mind. We evaluated worst-case scenarios and came up with a few countermeasures against them. The most serious threat we may face during a public data gathering is database pollution. We consider a database polluted when wrong data is indistinguishable from correct data and would change the expected result during our analysis (e.g., increasing false positives). First, we have to define what would be "wrong" data for our purpose and how such pollution may happen (intentionally or non-intentionally). Then, working to avoid the non-intentional pollution is the bare minimum. We also make intentional pollution difficult for unskilled people. However, protecting the data collection against skilled people is a challenging problem and is out of the scope of this work. We assume skilled people did not pollute our database to influence our results.

We consider the database polluted when two different recorded identities are the same machine (same stack of hardware and software). The interface to the data gathering has been made as simplistic as possible in order not to discourage users: we asked participants to provide an email address and to click on a button. Then, the data collection proceeded, displaying the percentage of data collected to the participant. So, what could go wrong? We have to consider every possible and problematic action a human might take:

- Doing the data gathering multiple times with the same email on the same machine.
- Doing the data gathering multiple times with different emails on the same machine.
- Stopping the data collection.
- Registering an identity but blocking the data collection. This could be unintentional due to some browser addons that are suspicious with canvas image extraction.

We prevent the first two bullets points by relying on a stateless attribute: we compute a browser fingerprint that should be unique to each browser. This fingerprint is transmitted with the provided identity. If the fingerprint matches another identity in the database, we display an error. Otherwise, the identity is accepted and a data collection begins. We could have used a cookie but it would be less protective against unskilled malicious users who might know that cookies are often used to "remember" and how to delete them. Finally, we asked users to complete a captcha to participate. Skilled malicious users may be slowed down or at the best, discouraged to fill our database with junk data. For the last two bullets, we discarded any user who have not finished the data collection. We did not notice any strange behaviour during the data gathering and, backed by our methodology, we are confident that our database has not been polluted.

Figure 4 shows the distribution of Operating System and Browser families from our participating users. The main operating systems and browsers are well represented with a social bias toward Unix-based system (due to a non-negligible fraction of colleagues and friends within the participants) and many different families have been collected, including smartphones, which is what we need to validate our result over the global software and hardware ecosystem.

4.3 CNN Architectures

Table 1 shows the two CNN architectures we use to evaluate our protocol. For each user, Arch 1 provides a learning phase under one minute, and Arch 2 provides a learning phase under three minutes with a docker virtual machine using 12 cores of Intel(R) Xeon(R) CPU E5-2670 v3 2.30GHz. It can be expected to be much faster with dedicated instruction sets on the hardware, which are available on the market, or by deploying the models in GPU clusters. These performances allow an identity provider to derive a user specific model in less than a minute for any registered user.

4.4 Results

4.4.1 CNN Accuracy. SWAT is designed to produce one CNN binary classification model for each registered user. The efficiency of our trained models is based on the accuracy of the chosen CNN architecture, as well as on the quality of the derived model that each user obtains. We may wonder what efficiency the CNN architecture achieves, and more importantly, what set of criteria might be used to define an efficient architecture. We define here the criteria that are important for our protocol:

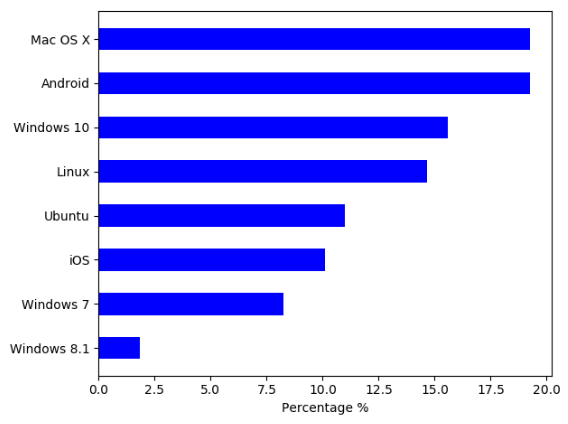
1. The model accuracy: how well does the CNN derived model perform for each user? We consider both the accuracy of the training set and the accuracy of the validation set.
2. The gap between best accuracy results and worst accuracy results.
3. The over-fitting: how large is the gap between trained results and validation results?
4. The number of epochs and the time for each epoch: how fast does the model learn?

The learning speed may be more or less important, depending on the targeted security level and on the amount of computational power that is available: high security requirements and high computational power availability push for the use of a more complex architecture, which leads to better results in terms of accuracy and best/worst performance gaps. We chose to work with fast architectures, showing how well we can perform at a moderate cost, without designing our solution around a specific hardware specification for our learning algorithm. This choice is guided by our main objective: offering a <realistic new authentication layer which can be deployed even for usage intensive applications.

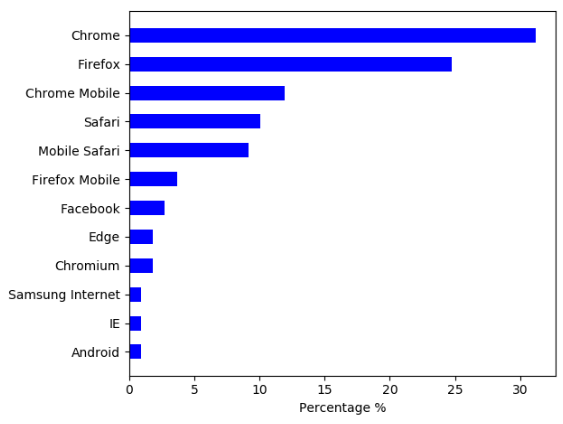
In Figure 5, we observe a median validation test reaching over 95% accuracy. The learning and validation sets contain canvas in the "yes" class and canvas in the "no" class chosen randomly from the database, and the accuracy is defined as the number of correct predictions over the total number of predictions. Figure 5b shows more over-fitting in the worst-case user (i.e., over all users and all runs), which indicates the need to collect more than 2000 canvas for that CNN architecture: because of the deep structure of the architecture, it may capture unique information about the training set which might not be representative of the user specific features.

4.4.2 True Positive Predictions. Figure 6 shows our classification's prediction results for new canvases during authentication. This figure shows the output of the authentication phase for legitimate authentications of each identity. The prediction result is not binary, it is a probability score for the canvas to belong to the 'yes' class. As can be seen on the figure, the second architecture shows a higher prediction distribution than the first one. We believe that there are two main reasons explaining this difference: we do not use max pooling in the second architecture and we have a deeper neural network. Indeed, max pooling is destructive (i.e., it discards information in favor of computational efficiency), and the deeper the network is, the more information it captures. However, those results alone are not enough to justify that the second architecture fits our authentication problem better. We also need to evaluate how a particular user performs against others. Ideally, we should obtain a 0 probability prediction for these situations.

4.4.3 False Positive Predictions. A high prediction result for true positive canvases would not be interesting if the CNN predicts also a high result of $user_i$'s canvas over $user_j$'s learned model. We need to evaluate this avenue to get a more precise idea of the effectiveness of this authentication method. Figure 7 shows CDFs of how well our CNNs perform for each user over all other users within our database. In Figure 7a, 'all' labeled lines compare our



(a) Operating System of each user of our database



(b) Browser of each user of our database

Figure 4: Percentage of observed OS family and Browser family in our data collection

	CNN		
Parameters	Arch 1	Arch 2	Search space
Optimizer	Adamax	Adamax	SGD, Adam, Adamax (default values)
Conv2D layers	3	17	3..20
Filters	4/layer	4/layer	4/8/16
Padding	valid valid same	valid (all)	valid, same
Dense	256 1	256 1	64/128/256/512 1
Flatten	yes	yes	-
Activation func	relu	relu	relu, tanh
Kernel init	Glorot normal	Glorot normal	-
Maxpooling	After each Conv2D	-	-
Pool Size	3x3 3x3 5x5	-	3x3 / 5x5
Batch size	256	256	64/128/256
Training epochs	100	100	50/100

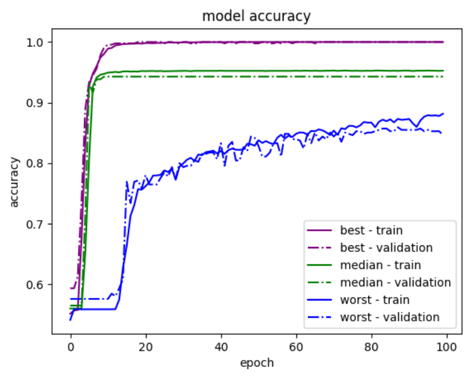
Table 1: Parameters choices for two architectures. Arch 2 is designed to offer better efficacy at a performance cost, which highlights the trade-off efficacy/performance of the various CNN architectures we could use

two architectures for all users, which shows that our CNN Arch 2 performs better as expected from the result of the previous section. Considering the 'all' labeled line, it gives us some intuition about the performance of this authentication method. Nevertheless, results about SWAT's performance in the real world may be better, because of two important points:

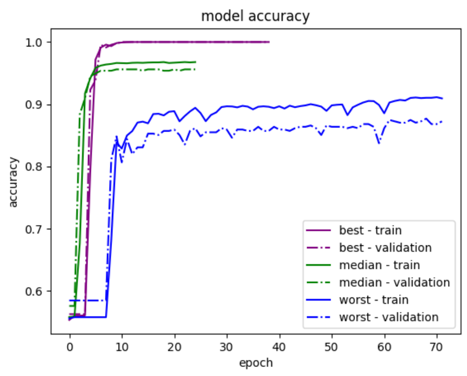
- Our data collection does not reflect the true distribution of hardware/software stacks which people own. Indeed, as we described in our Section 4.2, our data collection is biased toward Unix-based systems due to the author's social environment.
- Our results depends on the CNN used, and there is still strong potential for more efficient architectures.

This suggests that SWAT can achieve usable performances with a low training cost, and can potentially behave much better when deployed in a more realistic scenario by a company with the necessary computational resources.

We may also observe that users tend to obtain a higher score against other users running the same Operating System (Figure 7a) or the same Browser (Figure 7b). This result seems intuitive, as SWAT takes advantage of the various differences in the software/hardware stacks of users. When some parts of this stack are common, the CNN successfully extracts common features and may believe more easily that they are the same user. Another argument in favor of this interpretation can be observed with Mac OS X or iOS results: Due to less diversity in the hardware and software stack, Mac OS X



(a) Learning phase for Arch 1



(b) Learning phase for Arch 2

Figure 5: Model accuracy evaluation on CNN architectures

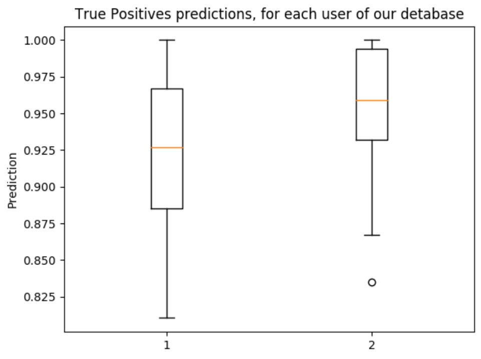


Figure 6: True positive prediction of CNN's architecture Arch 1 and CNN's architecture Arch 2

and iOS users are more likely to score high against each other (i.e., around 40% of our Mac OS X users score a high prediction against other Mac OS X users). We observe that this is less true as we move toward more complex CNN architectures, as showed by the results of the deeper CNN architecture.

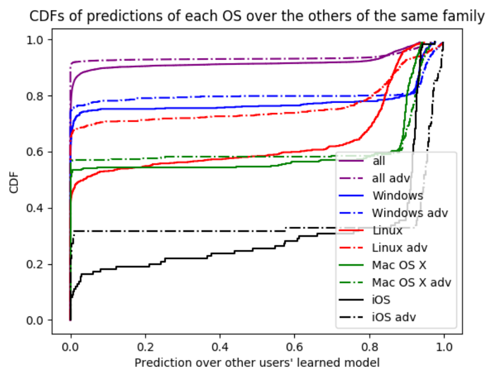
4.4.4 Accepting or Rejecting Authentication. An important question remains to be answered: how should we decide if an authentication is valid? The simplest and more straightforward technique would be to decide one global threshold which minimizes false positives and guarantees a true positive, based on our data. A second idea would be to use a threshold for each user, based on the result of their training phase. Indeed, since we derive one model per user, each of them produces a slightly different accuracy, as depicted in Figure 5. Therefore, having a model-based threshold to accept an authentication should give better results because some of the most accurate models can benefit from a higher threshold. A third and last idea would be to use, instead of a threshold, a validity interval for which we consider the authentication successful. Compared to the simple threshold, the validity interval would prevent $user_i$ from successfully authenticating as $user_j$ if $user_i$ obtains a much higher result.

Figure 8 shows the efficiency of our CNN architectures with ROC curves given two varying decision-making parameters: a global and unique threshold and a model-based interval for valid predictions. Since we know from the training phase which prediction value the valid user should obtain, we may use this information for the decision process to accept or reject an authentication attempt. This information is exploited with the model-based interval for valid prediction.

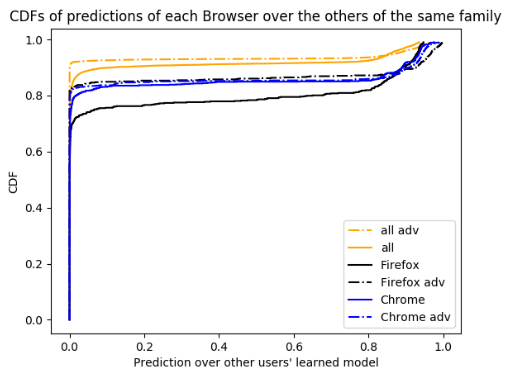
Figure 8a shows ROC curves for our two architectures given a global threshold varying from 0 to 1. For a TPR of 1, our CNN Arch 1 achieves $\approx 7,4\%$ of FPR and sets a global threshold to accept authentication to ≈ 0.81. The other deeper architecture Arch 2 achieves a TPR of 1 and $\approx 6\%$ of FPR, and sets a global threshold to accept authentication to ≈ 0.83. Figure 8b shows the result with a better heuristic to accept or reject authentication attempts. For each learned model, we accept an authentication attempt if the prediction is within an interval centered around the median value of the learning set's predictions. Figure 8b shows that this heuristic reduces the FPR to $\approx 5\%$. This improvement is motivated by two observations: first, by looking at Figure 7, we see that different users can score a high prediction for a given identity, even higher than the true identity itself. Hence, accepting authentication attempts simply based on exceeding a threshold is in fact allowing false positives. The second observation is that the prediction value is stable across several authentications of a same identity: if a training phase ends up giving a 0.85 prediction value to the canvas of one's identity, then this identity will score around 0.85 for each authentication. This stability characteristic allows a high TPR for small intervals.

4.5 Security Discussion

The current performance analysis of SWAT shows a non-negligible risk of a false positive. SWAT cannot hope to compete with the $\sim 2^{-128}$ security bounds provided by typical cryptographic tools, but this not its objective. Indeed, SWAT is not aimed to be a first-factor authentication, unless user friendliness is the critical property of the system (in such case, weak authentication is still better than

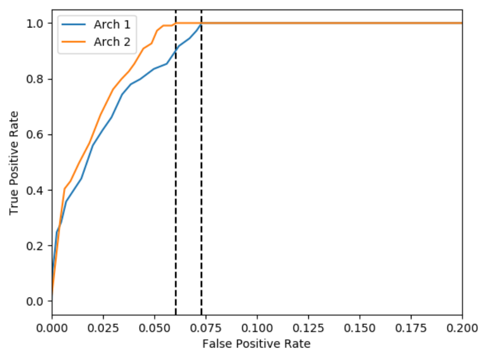


(a) Splitting users per OS families.

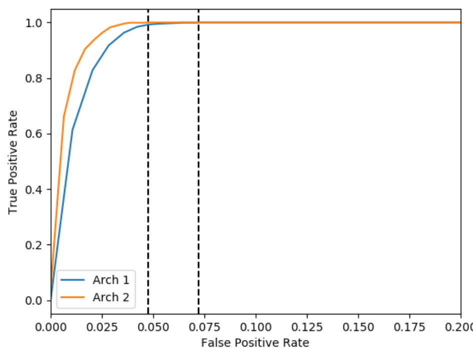


(b) Splitting users per Browser families

Figure 7: Prediction value over other registered users (i.e., $user_i$'s learned model evaluates $user_j$'s canvasses). "adv" labels refer to result for Arch 2, while the others refer for Arch 1



(a) Variation of a global threshold in interval [0, 1]



(b) Variation of the size of a model-based interval around the accuracy of each model predicted at the learning phase.

Figure 8: ROC curves for varying global threshold (left) and varying model-based interval (right). Vertical lines indicate the False Positive Rate when the True Positive Rate scores 1.

nothing). Rather, we expect SWAT to be useful in a multi-factors context where it would help to prevent direct access to the attacker if, for example, a password is leaked. In summary, SWAT enhances the security by a complementary threat resilience to existing authentication systems, without lowering the user experience if the identity provider implements a transparent usage of the protocol. We suggest in the next section a couple of potentially interesting usage scenarios.

5 POSSIBLE USAGE SCENARIOS

5.1 Web Authentication

A SWAT API could be built for various web frameworks. This work provides one implementation for the high-level Python web framework Django. With such an API, web developers could enhance the security of their application with multi-factor authentication, such as password + SWAT. With both authentication mechanisms covering different threat models, the benefits would be transparent for the users.

Another interesting use case would be to enhance the industry-standard protocol for third party authentication (Oauth2 [11]) with the ability to authenticate with SWAT. Identity providers such as Facebook, Google or Github are well-positioned and own the resource to develop and deploy our protocol in a transparent way for their users. For example, users may perform the learning phase when using the services offered by these Internet giants, and benefit from the increased security when connecting to others web services through Oauth.

5.2 IoT Authentication

Many IoT devices interact with smart phones and could benefit from our protocol inside dedicated applications as well. However, this would require the learning phase to be remote (not on the IoT device) and the learned model to be shared with the IoT device afterwards. In many settings, weak but user-friendly authentication is desirable and this protocol offers such an opportunity.

5.3 Game Launcher

The Game industry has known a huge growth since the free-to-play model was introduced. Hacking and frauds is now common against gamer accounts. One way to improve the current landscape would be the increase the authentication security in game launcher to prevent account theft and in-game content theft. Several game companies, like Blizzard, offer in-game bonuses when users improve their account security with a hardware token. Nevertheless, hardware tokens are not widely used. SWAT could be an interesting opportunity to deploy massively multi-factor authentication on game launchers, while not damaging the user-friendliness of their game.

6 RELATED WORK

Since Peter Eckersley's initial "device fingerprinting" analysis [4], work has spawned with various categories of goals. Laperdrix *et al.* [17] consider a few: *Client-side diversity* which tries to evaluate uniqueness of fingerprinting methods [4][17][9].

Fingerprinting adoption on the web and server-side scripts which evaluates the surface of the web under fingerprinting methods. Under this category, FPDetective [2] identified new scripts and techniques to fingerprint users. Acar *et al.* [1] investigated canvas fingerprinting, evercookies and use of cookie syncing on a large-scale study. Englehardt and Narayanan [6] pushed the analysis to 1 million websites, found sophisticated fingerprinting methods and showed how to crawl and analyze a huge amount of information in a reliable way.

Advanced solutions to collect additional fingerprintable attributes is the category focusing on developing new fingerprintng methods. Canvas fingerprinting [19] or leaking battery information through the HTML5 API [23] are good examples of work in this category, and show how much we deeply need to better understand our application environment to improve privacy.

Resilience to fingerprinting methods is a category of work where anti-fingerprinting strategies are investigated but could potentially break the accuracy of our authentication protocol [22][3][16]. As discussed in Future work, Section 7, defending against tracking while preserving beneficial usage of the output variation of the canvas API might be an interesting research direction.

Closer to what is performed in this paper is the category of *frictionless authentication* [24] where the users authenticate without performing any intentional authentication actions [20]. The usability obtained in frictionless authentication is granted through continuous monitoring of the claimed identity to spot any divergence from the expected user behavior. However, this raises some serious privacy concerns.

Our work, as far as we know, is the first attempt to use the output variation of some HTML5 API to the benefit of the user.

7 FUTURE WORK

7.1 Increasing drawing variations

Our work leverages on Mowery *et al.* [19] text and text's scene variation to distinguish users. Recent work by Gómez-Boix *et al.* [9] shows how to draw more complex canvas which would likely increase the accuracy of our authentication protocol. Smart utilization of the Canvas API to cover a broader surface of the software and hardware stack can be considered as a side problem that may contribute to various works, like our proposal, but also to the understanding of browser fingerprinting diversity.

7.2 Privacy please?

Different service could collude and match users as being the same identity. Future work could address this privacy issue and complicate such intersection attack given sets of canvas collected by different services. Such a privacy property seems however difficult to obtain with a stateless client side protocol. The current research for privacy-preserving browsing experience attempts to break the linkage between sessions of the output of HTML5 API [16]. This line of work would break our protocol and its benefit, but we could imagine opt-in per-service when the usage of distinguishable information is intended for a true user-centered benefit, which is in line with the current approach of browsers (e.g., Firefox opt-in to enable canvas extraction).

7.3 Active attackers

Our current analysis is based on canvasses produced in a genuine way. An adversary may proceed in a different way: when facing a challenge response, and assuming that he has access to previous responses to different challenges he may try to combine those responses in order to forge a fresh one. More generally, an adversary could use image processing software to create a canvas forging system. Investigating this direction would be interesting to evaluate the security of SWAT in the more demanding setting of active adversaries.

8 CONCLUSION

We presented SWAT: an authentication protocol with a focus on usability, server-side performance and ease of deployment. SWAT provides interesting security properties in a multi-factor authentication scenario, protecting against passive eavesdropping and replay attacks as well as defending against stolen credentials. We highlighted in the analysis the existence of a trade-off between authentication accuracy and resource endowment, and showed that we achieve a high TPR for a low FPR even for small resource endowment. We discussed SWAT's security usefulness in a multi-factors context and for various scenarios. A SWAT protocol is not bound the the strict use of the Canvas API. Any other API which can take an input, and output variations depending on user specific software or hardware would be interesting candidate to increase the authentication accuracy.

ACKNOWLEDGMENTS

We would like to thanks the anonymous reviewers for their feedbacks. We are also thankful to xsi.io's team for their valuable expertise and advises in Web technologies. This work was partially supported by the by the Brussels Region Research Funding Agency through the Innoviris/Secur'IT project C-Cure and by the Walloon Region through the "Plan Marshall 4.0" project Digitrans (convention number 7618).

REFERENCES

[1] Gunes Acar, Christian Eubank, Steven Englehardt, Marc Juarez, Arvind Narayanan, and Claudia Diaz. 2014. The web never forgets: Persistent tracking mechanisms in the wild. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 674–689.

[2] Gunes Acar, Marc Juarez, Nick Nikiforakis, Claudia Diaz, Seda Gürses, Frank Piessens, and Bart Preneel. 2013. FPDetective: dusting the web for fingerprinters. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*. ACM, 1129–1140.

[3] Peter Baumann, Stefan Katzenbeisser, Martin Stopczynski, and Erik Tews. 2016. Disguised chromium browser: Robust browser, flash and canvas fingerprinting protection. In *Proceedings of the 2016 ACM on Workshop on Privacy in the Electronic Society*. ACM, 37–46.

[4] Peter Eckersley. 2010. How unique is your web browser?. In *International Symposium on Privacy Enhancing Technologies Symposium*. Springer, 1–18.

[5] Steven Englehardt, Chris Eubank, Peter Zimmerman, Dillon Reisman, and Arvind Narayanan. 2015. OpenWPM: An automated platform for web privacy measurement. *Manuscript* (2015).

[6] Steven Englehardt and Arvind Narayanan. 2016. Online tracking: A 1-million-site measurement and analysis. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 1388–1401.

[7] Firefox. 2017. Prompt (w/ Site Permission) before allowing content to extract canvas data (Tor 6253). https://bugzilla.mozilla.org/show_bug.cgi?id=967895.

[8] Kunihiko Fukushima and Sei Miyake. 1982. Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition. In *Competition and cooperation in neural nets*. Springer, 267–285.

[9] Alejandro Gómez-Boix, Pierre Laperdrix, and Benoit Baudry. 2018. Hiding in the Crowd: an Analysis of the Effectiveness of Browser Fingerprinting at Large Scale. In *WWW 2018: The 2018 Web Conference*.

[10] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. MIT Press. http://www.deeplearningbook.org.

[11] Dick Hardt. 2012. *The OAuth 2.0 authorization framework*. Technical Report.

[12] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. 2017. Mask r-cnn. In *Computer Vision (ICCV), 2017 IEEE International Conference on*. IEEE, 2980–2988.

[13] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.

[14] Keras. 2018. Guide to the Functional API. https://keras.io/getting-started/functional-api-guide/.

[15] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*. 1097–1105.

[16] Pierre Laperdrix, Benoit Baudry, and Vikas Mishra. 2017. FPRandom: Randomizing core browser objects to break advanced device fingerprinting techniques. In *International Symposium on Engineering Secure Software and Systems*. Springer, 97–114.

[17] Pierre Laperdrix, Walter Rudametkin, and Benoit Baudry. 2016. Beauty and the beast: Diverting modern web browsers to build unique browser fingerprints. In *Security and Privacy (SP), 2016 IEEE Symposium on*. IEEE, 878–894.

[18] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324.

[19] Keaton Mowery and Hovav Shacham. 2012. Pixel perfect: Fingerprinting canvas in HTML5. *Proceedings of W2SP* (2012), 1–12.

[20] Mustafa A Mustafa, Aysajan Abidin, and Enrique Argones Rúa. 2018. Frictionless authentication system: Security & privacy analysis and potential solutions. *arXiv preprint arXiv:1802.07231* (2018).

[21] Vinod Nair and Geoffrey E Hinton. 2010. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*. 807–814.

[22] Nick Nikiforakis, Wouter Joosen, and Benjamin Livshits. 2015. Privaricator: Deceiving fingerprinters with little white lies. In *Proceedings of the 24th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 820–830.

[23] Łukasz Olejnik, Gunes Acar, Claude Castelluccia, and Claudia Diaz. 2015. The leaking battery. In *Data Privacy Management, and Security Assurance*. Springer, 254–263.

[24] Vera Rimmer, Davy Preuveneers, Wouter Joosen, Mustafa A Mustafa, Aysajan Abidin, Enrique Argones Rúa, et al. 2018. Frictionless Authentication Systems: Emerging Trends, Research Challenges and Opportunities. *arXiv preprint arXiv:1802.07233* (2018).

[25] Florentin Rochet. 2018. Github repository for project code. https://github.com/frochet/SWAT.

[26] Florentin Rochet. 2018. Interface used for data collection. https://perceval.elen.ucl.ac.be.

[27] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).

[28] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, Andrew Rabinovich, et al. 2015. Going deeper with convolutions. Cvpr.