Designed to be Broken: A Reverse Engineering Study of the 3D Secure 2.0 Payment Protocol

Abstract. 3 Domain Secure 2.0 (3DS 2.0) is the most prominent user authentication protocol for online payment using credit cards. 3DS 2.0 relies on risk assessment to decide whether to challenge the payment initiator for second factor authentication information (e.g., a passcode). The 3DS 2.0 standard itself does not specify how to implement transaction risk assessment. The research questions addressed in this paper therefore are: how is transaction risk assessment implemented for current credit cards and are there practical exploits against the 3DS 2.0 risk assessment approach. We conduct a detailed reverse engineering study of 3DS 2.0. for payment using a browser, the first study of this kind. We identify the data and decision making process that card issuers use in transaction risk assessment, for a number of different cards. We will see that different card issuers decide differently when to challenge payments. We will also demonstrate a practical impersonation attack against 3DS 2.0 that avoids being challenged for second factor authentication information, requiring no more data than obtained with the reverse engineering approach presented in this paper.

Keywords: Payment Systems · Credit Card Security · Reverse Engineering · User Authentication · Impersonation Attack

1 Introduction

In 2001 payment networks (Visa, MasterCard and Amex) introduced the 3 Domain Secure 1.0 (3DS 1.0) protocol [35]. 3DS 1.0 introduced user authentication, requiring payment initiators (customers) to prove their identity with static passwords. For instance, 'Verified by Visa' asked three characters of a registered password. 3DS 1.0 received criticisms for both security and usability reasons. Security was impaired because registering the password could not be guaranteed to have been done by the card owner, and phishing attacks on card data and passwords could also not be ruled out. However, the deciding drawback of 3DS 1.0 for merchants was 'lost sales', that is, customers who failed to complete the purchase because they cannot recall or refuse to go through the trouble of finding and entering the password [3, 23, 20, 16].

The European Commission proposed in 2015 the Payment Services Directive 2015/2366 (PSD II), a regulatory standard that asks card issuers within Europe to provide *Strong Customer Authentication* for *each* online payment transaction [13], very much like 3DS 1.0 would provide. The industry (card issuing banks, payment processors and online merchants) expressed concerns that the methods proposed in PSD II ignored the objectives of user-friendliness and argued that Strong Customer Authentication should be applied only to transactions

deemed 'high risk' in a *Transaction Risk Assessment* (TRA). After a six-month negotiation including over 200 payment industry stakeholders, Strong Customer Authentication in PSD II was augmented with Transaction Risk Assessment.

In October 2016, EMVCo (a consortium of card payment networks), revised 3DS 1.0 to include TRA, resulting in the current 3D Secure 2.0 protocol suite [10]. 3DS 2.0 provides two options: *challenged* and *frictionless* authentication. Challenged authentication is for purchases with a high risk and prompts an authentication challenge to the payment initiator. Frictionless authentication requires no additional authentication information and is meant for low-risk transactions. TRA sacrifices strict security requirements for usability–from a security perspective, it is 'designed to be broken'.

The 3DS 2.0 protocol does not specify how TRA should be implemented, apart from some generic guidance. Therefore, we present in this paper an in-depth investigation in existing 3DS 2.0 implementations, the first of its kind. We will show that transaction risk is determined from data collected through the payment initiator's browser, combined with transaction or network information (such as the transaction amount or IP address). The browser data acts as a 'fingerprint' of the user (see Section 2). In Section 4 we conduct an additional set of experiments with different transactions and from different locations to learn when the authenticator allows frictionless authentication. We will see that different card issuers implement TRA differently, with different issuers exhibiting considerably different risk appetite.

Our reverse engineering exercise uses five credit cards, from Visa as well as Mastercard, used at a number of different web sites. Experimental research with credit cards is challenging, for instance because of the possibility of blocked cards. It is therefore probably not surprising that the experimental research literature for online payment is relatively light, and that no studies on the scale of this paper exist. The five cards are representative for cards in general, in that the experiments generated similar fingerprint information. We note that all cards belonged to the authors, and ethics approval was obtained through regular processes of the authors' institution. Responsible disclosure through informing selected partners has taken place through our network of partners.

The design of 3DS 2.0 also suggests an obvious vulnerability, in that the authentication service may decide incorrectly not to challenge a payment. We will demonstrate an *impersonation attack*, in which a perpetrator impersonates a payment initiator, thus 'tricking' the authentication service into allowing a transaction to complete without being challenged for second factor authentication information, even for high transaction amounts. We will demonstrate two versions, one copying the fingerprint info to another machine that is configured arbitrarily, and one creating the same fingerprint on another machine with an identical configuration as the original machine. The impersonation exploit can be made into a practical attack if one manages to install malware that transports the fingerprint to the attacker, who can use it for a purchase impersonating the card holder (see Section 3 for details). This paper shows that such exploits can

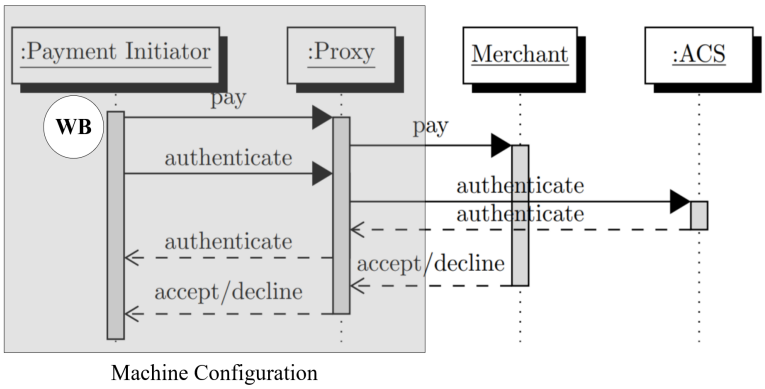


Fig. 1: Reverse engineering set-up, intercepting 3DS 2.0 transactions through a proxy.

be conducted by anyone who reverse engineers TRA in the manner of this paper, without requiring any additional knowledge about TRA.

2 Reverse Engineering Transaction Risk Assessment: Fingerprinting

3DS 2.0 specifies very little about how card issuers should implement Transaction Risk Assessment. To understand how merchants and card issuers assess the risk of consumer payments we therefore reverse engineer existing implementations.

2.1 Reverse Engineering System Set-up

Figure 1 shows the reverse engineering set-up. Within 3DS 2.0, a number of services and stakeholders are involved: the payment initiator using a browser, the merchant providing the check-out page at every purchase, and a set of services and servers for the authentication, termed the Access Control Server (ACS). The ACS maintains payment initiator's data which can be used to authenticate the cardholder during a purchase.

To intercept communication, we use the Fiddler proxy, which is available as open-source [32]. The proxy runs on the machine of the payment initiator (i.e., our own machine). We configure the machine's web browser (WB) to send HTTP(S) requests to Fiddler, which then forwards the traffic to the merchant or ACS. The responses are returned to Fiddler, which passes the traffic back to WB. When HTTPS decryption is enabled, the Fiddler proxy generates a self-signed root certificate and a matching private key. The root certificate is used to generate HTTPS server certificates for each secure site that is visited from WB.

Apart from intercepting the browser communication, we use two other techniques. First, using Fiddler, we challenge WB as if we were the merchant or the card issuer. Secondly, from Fiddler, we challenge the merchant as if the challenge was originating from WB. To handle ('tamper' in Fiddler terminology) a challenge, Fiddler provides a breakpoint function, which invokes a pause to

the communication. Once paused, we can tamper or edit the changes to the communication data.

In total, we used five test cards for our experiments, three Visa cards (C1-C3) and two MasterCard cards (C4, C5). To make sure that 3DS 2.0 does not have any machine identifiers pre-installed on the machine, we had a fresh installation of Windows 10 operating system and Chrome 59.x web browser.

The merchant web sites we used were all enabled with 3DS 2.0 checkout and were selected from the Alexa list of merchant web sites [2]. The 'Verified by (payment-network)' icon on the merchant web site indicates that it is 3D Secure enabled. To ensure that we have a representative sample of merchant web sites, we kept track of the ACS URL's to which our transaction were redirected. All 'Verified by (payment-network)' websites redirected us to the same ACS URL indicating that the implementation of 3DS is issuer based. For each test card, we made several legitimate transactions and recorded the complete checkout session for each transaction with Fiddler. We decided to stop making further transactions once authenticated by ACS using frictionless authentication. This ensures that the ACS trusts WB enough for frictionless authentication. We decoded the 3DS 2.0 transaction data as necessary and analysed the outcomes in detail.

2.2 3DS 2.0 Authentication Protocol

Figure 2 shows the transaction sequence for frictionless authentication over 3DS 2.0, collating 3DS 2.0 specification with transaction information extracted from Fiddler. The box labelled 'Tunnel (Customer,ACS)' represents the reverse engineered part of transaction visible from WB, while the transaction sequence steps for the rest of the parties are derived from 3DS 2.0 specifications.

In Figure 2 the customer initiates the payment in step 1 and in step 2 the merchant decides to trigger user authentication through 3DS 2.0. Step 3 and 4 set up the connection between payment initiator and ACS

Message 5 through 11 detail the interaction between browser and ACS, where the ACS retrieves the data from the browser used to assess the transaction risk. In step 6, the ACS sends JavaScript dfp.js to the browser and posts the results back in step 8. Note that dfp stands for device finger print, it aims at identifying the device by fingerprinting it, so that subsequent payment can be traced back to the same machine (and, therefore, more likely to the same payment initiator). If this is the first time the browser uploads the JavaScript, the ACS repeats the process in steps 9-11 to install persistent cookies (IDCookie) at the browser.

Hereafter, the transaction is processed according to the rules specified in EMV 3DS 2.0 specifications that states the 3DS Server to submit an Authentication Request (AReq) to the ACS. Transaction Risk Assessment is then completed in step 13, here resulting in frictionless authorization as indicated by No Challenge. The merchant can now submit an Authorization request (message 15).

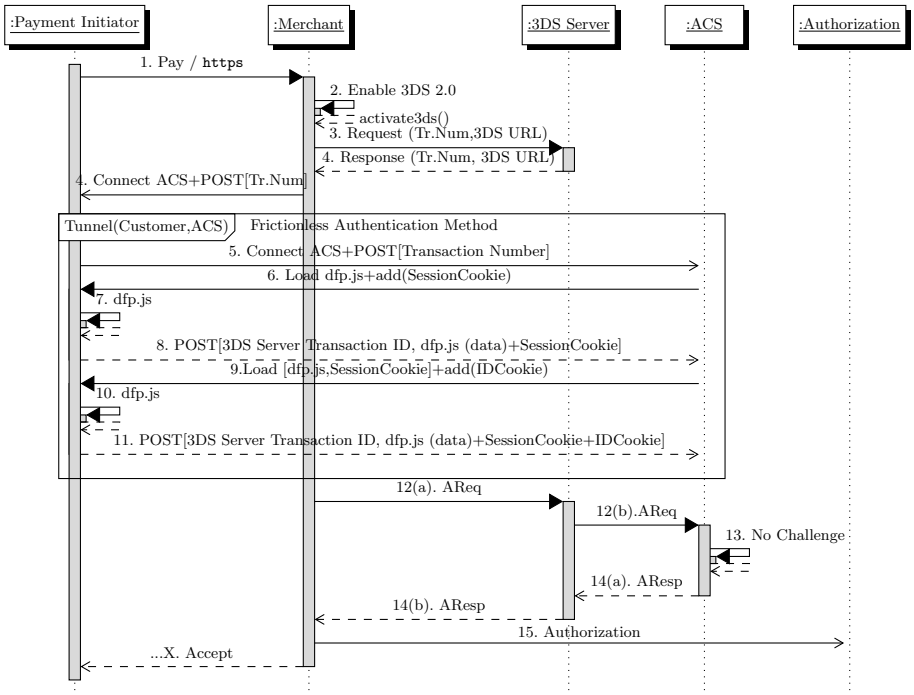


Fig. 2: Frictionless transaction sequence diagram.

2.3 3DS 2.0 Transaction Risk Assessment Data

The reverse engineering exercise shows how the ACS builds up a fingerprint of the payment initiator's machine. The ACS uses three pieces of information to establish a fingerprint, as discussed in this section:

1. the fingerprint information extracted from the browser using JavaScript
2. the 3DS 2.0 ID cookies fetched from the browser
3. the HTTP headers from payment initiator's browser forwarded by the merchant to the ACS

Fingerprint Data using JavaScript. The JavaScript fingerprinting scripts that we analysed contain functions to (i) collect browser-supplied information from the end-user device, and (ii) forward the collected data to the 3DS 2.0 server as a single Base-64 encoded string (the 3DS 2.0 specifications [10] requires all the data messages to be in Base-64 format). Table 2 in Appendix A shows an exhaustive list of device attributes from card `C1` to `C5` that are passed from WB to the ACS. The loading and execution of dfp.js by the ACS as a part of the checkout process is similar for all test cards that we used.

The data obtained is quite diverse, from browser and operating system information, to display, time, geo-location and some plug-in software information. The fingerprinting script obtains information that is part of HTTP headers through the `nav.userAgent()` and `test()` methods (see Table 2). The main method is

`deviceprint_browser()`, which gathers information about the browser and the operating system. With respect to geolocation, to the best of our knowledge, ACSs only use whether geolocation is enabled and the time zone of the machine (through `deviceprint_timez()`). It is likely that the ACS also uses URL and/or IP information as an indicator of location, but this is captured differently. Information about the hardware is obtained from `deviceprint_display()` and `window_information()`. Browser settings about tracking and advertisement preferences are provided by `DoNotTrack` and `Useofadblock`. Finally, `deviceprint_software()` and `flashscript()` provide information about specific hardware. In our experiments, only one ACS requested Flash information using `flashscript()`. To exchange the fingerprint information, dfp.js provides two more methods:

- `encode_deviceprint()` combines the collected data into a single string. It formats the string by removing whitespace, add delimiters and other characters as requires by the ACS.
- `asyncpost_deviceprint(url)` posts the data to the ACS URL. The data is converted to `base-64` before being sent as a form element to the ACS.

An example of resulting encoded device fingerprint is displayed in Figure 5 of Appendix A.

Cookies. We found three types of cookies installed by the ACS on our machines. These are also described in Table 2, bottom rows. Full cookies are displayed in Figure 6 of Appendix A.

- `Session_cookie`. Session cookie. The cookie is deleted after a user closes the session.
- `Test_cookie`. A test cookie with a name TESTCOOKIE and a value of Y was observed in exchanges during the transaction. This is set by the ACS server to determine if the user browser settings allowed cookies to be set.
- `IDcookie`. When the cardholder first enrolls into the 3DS 2.0 system, a token in the form of ID Cookie(s) is placed on the cardholder browser. The number of cookies installed varied from one to three. In all instances we found that these cookies have a validity of three years from the date of installation and also have an HTTP-only security tag. The HTTP-only tag on a cookie protects it from being accessed by cross-domain websites.

Data Passed from Merchant to ACS. Data passed by the merchant in AReq message (step 12 of Figure 2) contains elements that identify payment initiator browser configuration. For instance, Table A.1 in the EMV 3DS 2.0 specifications [10], suggests merchants to pass browser accept headers, language, screen details and user agent in the AReq message. The browser configuration helps the ACS to render a correct iframe for the cardholder device and may be used by the ACS to compare the information passed with `dfp.js`. To inspect the methods by which the merchant collects data to frame the AReq message, we referred to the merchant developer guides from payment networks Visa [36] and MasterCard [22] and payment service providers like PayPal [26], which suggest to use the HTTP headers passed on by the merchants during checkout as a part of WB's authentication data.

2.4 Discussion of 3DS 2.0 Implementations.

There exist a number of notable differences between different implementations of 3DS 2.0. These differences can be categorized as follows:

1. difference in the use of 3DS protocol version
2. difference in transporting the device fingerprint: obfuscated versus plain-text
3. difference in amount of data collected as a fingerprint: JavaScript based versus HTTP headers and cookies only.

Difference in the use of 3DS protocol version. We observed that the ACS associated with card `C2` adds a layer of frictionless authentication over the 3DS 1.0 protocol. As opposed to 3DS 2.0, the browser collects and submits the AReq message with the transaction identifier, following the 3DS 1.0 specification. The ACS installs and collects the fingerprint data from the browser. Similar to 3DS 2.0 frictionless authentication, if this is the first 3DS 1.0 transaction from the machine, the ACS repeats messages to install IDCookie. Hereafter the transaction is processed according to the 3DS 1.0 specification. The ACS decision (to not challenge) is added to the ARes which is then forwarded to the merchant via the browser. Comparing the frictionless authentication of 3DS 1.0 and 3DS 2.0, both of these protocols capture static fingerprint data in base-64 encoded format and use HTTP-only IDCookies for TRA.

Difference in device fingerprint implementation. In two cases (`C2` and `C5`) we noticed that code obfuscation techniques were applied to make the JavaScript difficult to read and analyse. However, obfuscated codes has certain general limitations, in that, it is an encoding technique (not encryption) and needs to make sure that the code does not loose its functionality when executed over the system. The 3DS 2.0 device fingerprint JavaScript can still be run to obtain base-64 device fingerprint values.

Additionally, code obfuscation is a technique that has long been used by malware writers to hide their malicious code. Therefore there are plethora of security tutorials and freely available security tools designed to de-obfuscate JavaScript. The most reliable de-obfuscater that we discovered for our research is available as open source from Intelligent Systems Lab, Zurich [18].

Difference in amount of machine data collected. Although Table 2 shows an exhaustive list of all the data elements collected by the fingerprinting scripts and HTTP headers, the amount of data collected by each implementation of the JavaScript varies substantially. Some of the card issuers have no device fingerprinting JavaScript implemented at all. For example the card issuer of `C3` implements frictionless authentication over 3DS 1.0 and only relies on the data received in the AReq message.

As a final note, the 3DS 2.0 protocol also defines an enrolment phase during which the card issuer collects the fingerprints from the card issuer computer and signs the fingerprint data to create ID cookies. The card holder computer is then 'tagged' through the usual cookie mechanism with these ID cookies. This

enrolment phase is imperfect, in that it cannot be determined if the payment initiator who enrols a certain card is a legitimate user of the card.

3 Impersonation Attack

In this section we device a realistic impersonation attack, where an attacker uses obtained data described in the previous section and avoids being challenged for a second factor of authentication information. We first describe the precise attack model in Section 3.1, and then explain in Section 3.2 how the attack can be implemented, particularly related to obtaining the data. We carried out a number of experiments with different machines to demonstrate that the impersonation attack indeed succeeds, as we will describe in Section 3.3.

3.1 Attack Model

The objective of the attack is to use the credit card of another party to successfully complete an online purchase, despite the fact that the merchant uses 3DS 2.0. We assume that the attacker has no manner in which it could respond successfully to a challenge for a second factor of authentication information. Therefore, the objective of the attacker is to avoid a challenge and be allowed to complete a frictionless transaction. We consider the attack successful if an attacker avoids being challenged in situations the ACS actually should challenge.

To succeed, the attacker needs to obtain the credit card details, the cookies and the fingerprint data used for Transaction Risk Assessment, as described in the previous section. We do not assume any insider administrative access privileges of the attacker, neither at the payment initiator's machine nor at any of the 3DS 2.0 services. The attack assumes a perpetrator manages to install malware or plug-in that collects the necessary data from the payment initiator's machine, which includes running the JavaScript fingerprinting scripts–we will argue in the next section that that is not far-fetched. Shipping this data to the attacker allows the attacker to impersonate the cardholder's identity by crafting its 3DS 2.0 authentication data to be identical to that of the payment initiator.

3.2 Attack Implementation

The attack implementation needs to complete two stages: (1) obtaining the card and transaction risk assessment data, and (2) using the card and transaction risk assessment data.

Obtaining Card and Transaction Risk Assessment Data. In this stage, the attacker needs to obtain credit card details and machine fingerprint data (including cookies). There is a variety of reasons why this can only be done through a Man in the Browser.

A challenge is that the ID cookies (see Section 2.3) are `http`-only protected, that is, they cannot be read by any cross-domain web pages or through JavaScript. Browsers allow access to `http`-only cookies to extensions (including malware) because extensions are considered "trusted" once installed, whereas regular JavaScript

is not. Cross-site scripting (XSS) [5, 4, 33], in which a script from a web site different than the merchant or 3DS 2.0 server attempts to access information such as cookies, is therefore not possible.

The most basic approach to obtain the required data is a browser plug-in that can sniff the browser communication to steal http-only cookies, record keystrokes to steal user payment data and execute device fingerprint JavaScript to capture the device fingerprints. More advanced malwares have such features, and are commonly available at [21][38], see for instance the ZeUS, SpyEye, Dridex and Tinba malwares. Once such malware is installed, it has an ability to obtain card transaction data for a purchase, the associated transaction risk assessment data described in the previous section, as well as the http-only cookies [12, 19, 34]. Malware SpyEye, for example, gets into a browser by prompting them to install a pdf reader or a flash player plug-in. Once into the browser, it updates itself as needed to configure fake entity certificates into the browser storage, record keystrokes, sniff the browser communication, records browser sessions and even capture screen shots [31, 15].

Using the Obtained Card and Transaction Risk Assessment Data. The task in exploiting the obtained data is to impersonate the card holder in the attacker's browser. The attacker copies the cookies to their own browser, and initiates a transaction with the merchant of choice, even if the merchant uses 3DS 2.0. It also receives credit card details and machine fingerprint data, per the above. At payment, the attacker creates or replays the correct responses in the protocol of Figure 2. Since there is no randomness in the fingerprint data, the same string of dfp.js data and HTTP headers obtained from the payment initiator's machine can be replayed on the attacker's machine using Fiddler (if required). To tamper the data, fiddler breakpoints are added whenever the merchant and the ACS connect to the attacker's browser.

3.3 Attack Demonstration

The demonstration of the attack aims to identify if it indeed is possible to impersonate from a different machine a legitimate payment initiator. In this demonstration we use the data obtained from machine `M1`, using the experiment set-up from Figure 1. We randomly selected a merchant with 3DS 2.0 enabled checkout and repeated transactions using all test cards `C1` to `C5` until `M1` was trusted enough for frictionless authentication. The payment sessions made from `M1` were recorded by the Fiddler proxy and were reused on a differently configured machine `M2`. We also show how a different machine `M3` that is identically configured generates the same fingerprint. We note that `M2` and `M3` were on networks different from `M1`, so that the IP source address is different.

The approach behind our experiments is as follows. We conduct the experiment for the five credit cards mentioned. First, we ran an experiment to verify that transactions from the differently configured machine `M2` are indeed challenged if one only enters card information (and does not impersonate the card holder with the risk assessment data). This verification was successful in all of the cases except for card `C1` where lower value transactions below £10 were approved (we will get back to this in the next section). Then, we ran an experiment

in which we used the obtained transaction risk assessment data to impersonate the card holder, to see if we were allowed to complete the purchase unchallenged, i.e., in frictionless mode. We initiated transactions where we selected products with values ranging between a £1 to £300, on an online merchant that uses 3D Secure 2.0 at checkout.

We were successfully able to execute the attack for all our test cards (`C1-C5`), in that the transactions were approved without any challenge by the card issuing bank's ACS. Interestingly, only for test card `C5`, the card issuer ACS issued challenges when the value of transaction reached above £200 (a typical transaction threshold set for frictionless authentication).

We ran a second experiment, using a different but identically configured machine `M3`, with the same hardware and software as `M1`. In so doing, we wanted to see if different machines that are configured identically generate identical Fingerprint data. This is to simulate a scenario where an attacker is unable to obtain the device fingerprint data but was able to get the ID cookies. In all cases, the transactions were allowed to go on without being challenged. Close inspection of the data that `M3` sent to the merchant and ACS revealed that the transaction risk data was essentially identical for `M1` and `M3`.

Reflection. For consumers it would be important to know how merchants and card issuers respond if the above attack took place. To that end, we communicated with the card issuing banks to understand how it would react if we were to report the fraudulent transactions that were made from the attacker machine. The card issuer for `C3` asks cardholders to identify some previous transactions made from the victim's machine and would not register the transactions made from attacker machine as fraud. The card issuer for `C3` also blocks and re-issues a new payment card to the card holder. However, in two cases (`C4` and `C5`), the card issuer argued that the transactions must have originated from the actual card holder's machine. They argued the card holder is trying to perform a 'friendly fraud', and so is denied a refund of any reported losses. This paper shows that this conclusion is not necessarily correct.

4 Reverse Engineering Transaction Risk Assessment: Decision-Making

Section 2 established which data 3DS 2.0 implementations used in their transaction risk assessment, and we showed that with that data alone, one can execute an impersonation attack. However, this does not yet provide us with full understanding of the way risks are being assessed by the ACS. First, the ACS may use additional sources of data, for example, it may use header info from the protocol stack such as the IP source address or some other data about the card holder available from the card issuer. Secondly, the ACS will set certain rules about when to invoke a challenge. These rules will stipulate which fingerprint data to consider, and specifies bounds on data outside which the transaction will be challenged (e.g., a limit for the transaction amount).

There are number of questions of interest motivating further re-engineering of the risk assessment approach. First, it provides information about which variants

Table 1: Experiments with and results for cards C1 and C2

Transaction number	Scenario	Machine data	Cookie ID	Value (£)	Region	Website	Card	Challenged?	Transaction status	Blocked
T1	S1	✓	✓	10	✓	W1	C1	×	Approved	×
T2							C2	×	Approved	×
T3						W2	C1	×	Approved	×
T4							C2	×	Approved	×
T5	S2	✓	✓	309	✓	W1	C1	×	Approved	×
T6							C2	×	Approved	×
T7						W2	C1	×	Approved	×
T8							C2	×	Approved	×
T9	S3	✓	✓	10	×	W1	C1	×	Approved	×
T10							C2	✓	Declined	×
T11						W2	C1	×	Approved	×
T12							C2	×	Approved	×
T13	S4	✓	✓	309	×	W1	C1	×	Approved	×
T14							C2	✓	Declined	✓
T15						W2	C1	×	Declined	×
T16							C2	✓	Declined	✓
T17	S5	×	×	10	✓	W1	C1	×	Approved	×
T18							C2	×	Approved	×
T19						W2	C1	×	Approved	×
T20							C2	✓	Declined	×
T21	S6	×	×	309	✓	W1	C1	✓	Declined	×
T22							C2	✓	Declined	✓
T23						W2	C1	✓	Declined	×
T24							C2	✓	Declined	✓
T25	S7	×	×	10	×	W1	C1	×	Approved	×
T26							C2	✓	Declined	×
T27						W2	C1	×	Approved	×
T28							C2	✓	Declined	×
T29	S8	×	×	309	×	W1	C1	✓	Declined	×
T30							C2	✓	Declined	✓
T31						W2	C1	×	Declined	✓
T32							C2	✓	Declined	✓

of the impersonation attack would succeed and thus allows us to assess the security and risks behind online payment. Secondly, it serves as a suggestion for a possible methodology to assess consumer implications of Transaction Risk Assessment. TRA shifts liability to the card issuer but nevertheless still exposes consumers to possible distress when an impersonation attack is carried out. Arguably, it would be in the interest of the public if there is visibility in the implementation of Transaction Risk Assessment. The re-engineering experiments in this section demonstrates how to provide such visibility.

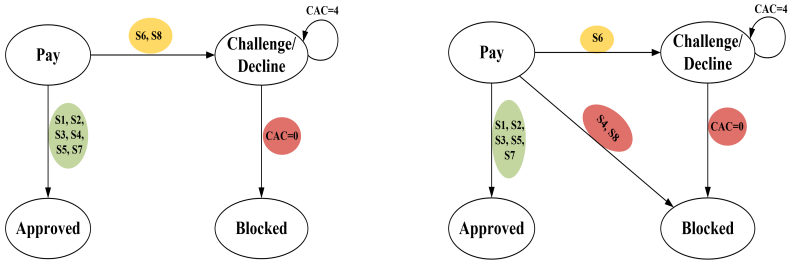
The experiments in this section obtain responses from the ACS for transactions in 8 different scenarios. These scenarios provide all combinations of the following three features:

1. submitting the machine data and IDCookie or not (from Section 2.3 and 2.3)
2. submitting different transaction values
3. submitting transactions from different regions

Table 1 shows selected results from our experiments on two test cards C1 and C2. Our set-up was identical to Section 3, with data obtained from machine M1 used on an alternative machine M2. Payments were initiated on two merchant websites (W1 and W2) that enforce 3DS user authentication. W1 is a web merchant local to the country where the victim card is issued and W2 is an overseas merchant for a victim's card.

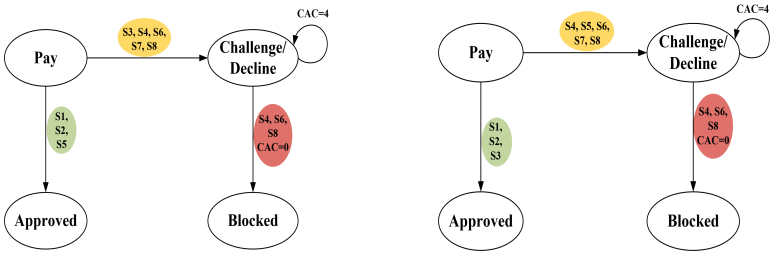
The rows give the various scenarios. For instance, Scenario S1 copies the machine data and the ID Cookie, for a low value transaction, within the region. With respect to the region, experiments for C1 and C2 were made from UK and Germany. Region (✓) indicates the transaction attempts were made from same country.

We see from Table 1 that different card issuers make different risk tradeoffs. In particular, the card issuer of C1 allows more frictionless authentication, whereas the card issuing bank for C2 challenges the payment initiator more often. Comparing transaction T4 and T10 for C2 we see that C2's card issuer challenges every transaction if the web merchant is in a different country. Table



(a) `C1` on `W1` (b) `C1` on `W2`

Fig. 3: Summarising `C1`'s risk assessment outcomes over merchants `W1` and `W2`



(a) `C2` on `w1` (b) `C2` on `W2`

Fig. 4: Summarising `C2`'s risk assessment outcomes over merchants `W1` and `W2`

1 also shows that cards are generally treated more harshly, when transactions are made from different regions. For instance, when transactions were made from different country and machine data is corrupted there is more likelihood of being challenged and transaction being declined (as opposed to transactions when initiated from the country local to the card issuer).

Figure 3 and Figure 4 summarize the findings of Table 1. The 'states' are phases in the 3DS 2.0 transaction, where `Pay` indicates initiating payment, while the other refer to possible outcomes, either approved, challenged/declined or blocked. Note that for our purposes we do not have to differentiate between challenge and declined, they both imply that the transaction has not gone through as frictionless. The arcs are labelled with the scenario given in the second column of Table 1. `CAC` stands for challenge limit counter, which counts down from the limit to zero. Here, the limit is 4, and at the fifth attempt the card is blocked. For an impersonator, Figure 3 and Figure 4 serves as a reference map in case where more card details are stolen belonging to `C1` and `C2` card issuer.

5 Discussion of Card Payment Systems Security

The problem of authenticating cardholders in the online payment system is exacerbated by the desire to cause minimal friction during the checkout. The introduction of 3DS 2.0 addresses this security/usability challenge through the use of Transaction Risk Assessment, and it is clear that the industry strongly favours

such risk based approaches, given that in the US about 75% of the card issuers have adopted risk-based authentication [7]. However, as we have seen in this paper, the remaining security bottleneck is the secure storage and transfer of machine authentication data and `http`-only cookies from the customer machine to the authentication service.

Once 3DS 2.0 is common and authorization-only transactions can no longer be exploited, the impersonation attack presented in this paper is potentially attractive for perpetrators. Its net effect would be that perpetrators can use stolen 3DS 2.0 frictionless authentication data in online shops without the cardholder being negligent, exactly as was the case with authorization-only systems before the introduction of 3D Secure. The attack does not require to synchronize fraudulent purchase with that of an unwitting customer (as a relay attack would). Malware could easily be designed to sniff the 3DS 2.0 transaction data and later forward it to the attacker server. In fact, there are a number of such open source browser extension available and installed be thousands of browsers, e.g., HTTPWatch [17] and LiveHTTPHeaders [11]. Other developments, such as FraudFox [37], are also cause of concern. FraudFox aims to make it faster and easier to change a browser's fingerprint to one that matches that of a victim, for instance through profile generator scripts.

Attempts to complicate executing the attack through JavaScript obfuscation, as some implementations do, cannot be expected to be of much help. There exist several tools and tutorials on the Internet which can be useful to re-establish the original data and script obfuscation is therefore far from sufficient. More helpful is the manner in which cookies are stored in the observed implementations. All ID cookies we discovered were secure enabled, which means the cookies are only passed on secure connections (HTTPS). Secondly, the cookies were tagged `http-only`, which implies that the cookie is not readable to JavaScript. This prevents the cookies from being accessed by the cross-domain websites, i.e., prevents cross-site scripting attacks `(XSS)`. Nevertheless, cookie storage in browsers remains non-secure unless the machine uses secure storage.

Technologically, an obvious solution for secure transfer would be to use private/public key approaches to encrypt and sign messages between the payment initiator and the 3DS Server. However, for such a solution to gain acceptance would require a separate trusted secure storage environment for cryptographic keys and certificates. The payment industry standards [27, 28] require payment credentials, including keys and certificates to be stored in 'Tamper-Resistant Security Module,' which is defined as the set of hardware, software, firmware, or some combination thereof that implements cryptographic logic or process (including cryptographic algorithms and key generation) and is contained within the cryptographic boundary. Today's computer systems and their software systems are not provably secure enough. This issue has come up before, when Google first introduced Android pay with the concept of Host Card Emulation with Android KitKat 4.4 [14] in 2014. The key storage security model for Host Card Emulation was software controlled and contained the threat that an attacker

may compromise the mobile OS to steal the credentials. This approach was therefore not found suitable to host EMV payment applications [1].

6 Related Work

This section details the comparison of card payment protocols and the security technologies they utilize. The section also highlights reported attacks on card payments that are made possible when any security feature is not included in the protocol. It would go too far to discuss the technologies and protocols in all detail, but we provide a summary discussion of the salient points.

Solutions for Card Present. This category corresponds to payments when the card is physically present. With magnetic stripe cards, data integrity and card authentication (confirming the identity of the card) features were not placed on the actual card itself. The data stored in a magnetic stripe is static and is kept in plain text which made magnetic stripe cards vulnerable to identity theft attacks [23], cardholder impersonation attacks [24] and card cloning attacks [6].

EMV extended the features of smart cards which provided a secure, "tamper proof", storage for the card's private cryptographic keys. The Chip and Pin protocol defined by EMV makes use of RSA public key infrastructure in three variants. The Static Data Authentication (SDA) card has a static signature which is generated by the issuer signed by using the issuer's private key, and written to the SDA card during manufacture. However, static signatures are used to approve every transaction, which makes SDA cards vulnerable to cloning attacks [6][25]. Dynamic Data Authentication (DDA) payments on the other hand generate a unique 'challenge-response' RSA signature (SDAD) for each transaction, including a nonce. Combined Data Authenitcation (CDA) improves upon DDA by encoding the Application Cryptogram into the signature rather than the transaction data. This makes DDA and especially CDA highly robust against any form of attack.

EMV contactless provides convenience to the customer by authenticating the card instead of actually prompting the cardholder to approve the transactions [9]. Fast DDA (fDDA) and CDA (fCDA) are enhanced versions of DDA and CDA of EMV chip and PIN, excluding the cardholder authentication methods from the protocol. Both DDA and SDA offer protection against known attacks on the payment system, however, each DDA and SDA enabled transactions would require the cardholders to prove their identity, thus adversely affecting usability. This was further addressed with an enhanced versions of fDDA and fCDA in EMV contactless [8].

Solutions for Card Not Present. If the card is not present, the situation is very challenging, as we have seen in this paper. As discussed in the introduction, the complications associated with the implementation of the 3DS 1.0 protocol made it possible for attackers to bypass its security features and perform identity theft attacks [23][16]. Chip Authentication Programme (CAP) and Transaction

Authentication Numbers (TAN) [30][29] are two token generation technologies that consumers use to produce the answer to a challenge from the authorization system. Typically, this is done with a little machine that reads a credit card and/or uses a PIN to generate a response to a challenge. These are increasingly commonly provided by banks, but in many cases are limited to payments through banking transactions.

In conclusion, different payment protocols have been developed for different purposes. Satisfactory solutions find a successful combination of usability and security, and also manage the exposure to risk were something to go wrong. For instance, transaction limits on contactless cards as well as the frictionless 3DS 2.0 payment limit both manage the risk by limiting loss exposure of consumers. Not surprisingly, sound approaches challenge for a second factor information, through a PIN such as in Chip & PIN as well as Challenged Authentication in 3DS 2.0 or using token generators such as in CAP and TAN. However, these do not satisfy the usability wishes of merchants, leaving consumer with systems such as 3DS 2.0 that are designed to allow less secure payments and therefore inherently (and by design) expose consumers and card issuers to fraud.

7 Conclusion

This paper presents the first sizeable experimental study of real-life implementations of 3DS 2.0. Through a reverse engineering study, we map out the transaction sequences for frictionless transactions. In most implementations we encountered, the payment initiator's machine is fingerprinted through JavaScripts, except for the implementation based on 3DS 1.0. In our experiments we obtained further insights in the decision making of the authorization service, experimenting with transaction amounts and the region from which payment was initiated. We found that card issuers differ in terms of their risk appetite, with some issuers considerable more liberal in allowing transaction to proceed unchallenged.

We also demonstrated an impersonation attack against 3DS 2.0, using only data that is available from a reverse engineering exercise such as described in this paper. This impersonation attack is practically feasible and exploits that fingerprinting information from the payment initiator's machine can be recreated by malware or plug-ins, if installed on that machine. This exploit demonstrates the vulnerability of credit card based payment using browsers, compared to the more sophisticated security of mobile payment solutions.

A key question for the regulator is whether it was justified to allow risk assessment based approach to online payment security as result of the PSD II negotiations. A complete answer to that question would require insight in a variety of factors, including technological feasibility and acceptance, ease-of-use, liability, as well as vulnerabilities and threats. In addition, one would need deeper insight into the specifics of the risk assessment carried out by the card issuer. However, the reverse engineering approach introduced in this paper provides an interesting set of tools to find out how risk assessment is implemented and for the regulator to assess whether the resulting decisions are in the interest of customers.

References

1. Ahmad, Z., Francis, L., Ahmed, T., Lobodzinski, C., Audsin, D., Jiang, P.: Enhancing the security of mobile applications by using TEE and (U)SIM. In: 2013 IEEE 10th International Conference on Ubiquitous Intelligence and Computing and 2013 IEEE 10th International Conference on Autonomic and Trusted Computing. pp. 575–582 (Dec 2013). https://doi.org/10.1109/UIC-ATC.2013.76
2. Alexa: Alexa - Top Sites by Category: Business/E-Commerce (2018), `https://goo.gl/V52tcs`
3. Ali, M.A., Arief, B., Emms, M., van Moorsel, A.: Does the online card payment landscape unwittingly facilitate fraud? IEEE Security and Privacy **15**(2), 78–86 (2017)
4. AOWASP: Cross-site scripting (XSS) OWASP (2018), `https://goo.gl/x54ner`
5. Barth, A., Caballero, J., Song, D.: Secure content sniffing for web browsers, or how to stop papers from reviewing themselves. In: Security and Privacy, 2009 30th IEEE Symposium on. pp. 360–371. IEEE (2009)
6. van den Breekel, J., Ortiz-Yepes, D.A., Poll, E., de Ruiter, J.: EMV in a nutshell (2016)
7. CardinalCommerce: Use of consumer authentication in ecommerce, annual survey 2017: The fraud practice (2017), `https://goo.gl/z2mByt`
8. Emms, M., Arief, B., Freitas, L., Hannon, J., van Moorsel, A.: Harvesting high value foreign currency transactions from EMV contactless credit cards without the PIN. In: Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security. pp. 716–726. CCS '14, ACM, New York, NY, USA (2014). https://doi.org/10.1145/2660267.2660312, `http://doi.acm.org/10.1145/2660267.2660312`
9. Emms, M., Arief, B., Little, N., van Moorsel, A.: Risks of offline verify PIN on contactless cards. In: Sadeghi, A.R. (ed.) Financial Cryptography and Data Security. pp. 313–321. Springer Berlin Heidelberg, Berlin, Heidelberg (2013)
10. EMVCo: 3D Secure 2.0 (2017), `https://goo.gl/d1ksLf`
11. E.solutions: Live HTTP Header (2018), `https://www.esolutions.se/`
12. Etaher, N., Weir, G.R., Alazab, M.: From ZeuS to ZitMo: Trends in banking malware. In: Trustcom/BigDataSE/ISPA, 2015 IEEE. vol. 1, pp. 1386–1391. IEEE (2015)
13. EU Council: Directive (EU) 2015/2366 (2015), `https://goo.gl/psyvps`
14. GoogleAndroid: Android pay (2014), `https://www.android.com/pay/`
15. Harshit Nayyar: Clash of the Titans: ZeuS v SpyEye. SANS Institute InfoSec Reading Room (2010), `https://www.sans.org/reading-room/whitepapers/malicious/clash-titans-zeus-spyeye-33393`
16. Herley, C., Van Oorschot, P.C., Patrick, A.S.: Passwords: If we're so smart, why are we still using them? In: International Conference on Financial Cryptography and Data Security. pp. 230–237. Springer (2009)
17. HTTP Watch: HttpWatch 11: HTTP Sniffer for Chrome, IE, iPhone and iPad (2018), `https://www.httpwatch.com/`
18. Intelligent Systems Lab: JS NICE: Statistical renaming, Type inference and Deobfuscation (2018), `http://jsnice.org/`
19. Kim, D., Kwon, B.J., Dumitraş, T.: Certified malware: Measuring breaches of trust in the windows code-signing PKI. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. pp. 1435–1448. CCS '17, ACM, New York, NY, USA (2017). https://doi.org/10.1145/3133956.3133958, `http://doi.acm.org/10.1145/3133956.3133958`

20. King, R.: Verified by Visa: bad for security, worse for business - Richard's Kingdom (2009), https://goo.gl/NgUUvn
21. MalShare: Malware Repository for Researchers (2018), https://malshare.com/
22. Mastercard: Merchant SecureCode implementation guide (2014), https://goo.gl/DyQ7Jb
23. Murdoch, S.J., Anderson, R.: Verified by Visa and MasterCard SecureCode: Or, how not to design authentication. In: Proceedings of the 14th International Conference on Financial Cryptography and Data Security. pp. 336–342. Springer Verlag (2010)
24. Murdoch, S.J., Anderson, R.: Security protocols and evidence: Where many payment systems fail. In: International Conference on Financial Cryptography and Data Security. pp. 21–32. Springer (2014)
25. Murdoch, S.J., Drimer, S., Anderson, R., Bond, M.: Chip and PIN is Broken. In: 2010 IEEE Symposium on Security and Privacy. pp. 433–446. IEEE (2010). https://doi.org/10.1109/SP.2010.33
26. PayPal: PayPal Pro - 3D secure developer guide (2018), https://goo.gl/7mPWWt
27. PCIDSS: Payment card industry (PCI) data security standard requirements and security assessment procedures (2016), https://goo.gl/PNSEq3
28. PCISCC: Payment card industry (PCI) hardware security module (HSM) security requirements (2009), https://goo.gl/JQKH3T
29. RedTeam Pentesting: Man-in-the-Middle Attacks against the chipTAN comfort Online Banking System. Tech. rep. (2009), https://www.redteam-pentesting.de/publications/2009-11-23-MitM-chipTAN-comfort_RedTeam-Pentesting_EN.pdf
30. RedTeam Pentesting: New banking security system iTAN not as secure as claimed. Tech. rep. (2009), https://www.redteam-pentesting.de/en/advisories/rt-sa-2005-014/-new-banking-security-system-itan-not-as-secure-as-claimed
31. Sood, A.K., Zeadally, S., Enbody, R.J.: An empirical study of HTTP-based financial botnets. IEEE Transactions on Dependable and Secure Computing **13**(2), 236–251 (2016)
32. Telerik: Fiddler web debugging tool (2018), https://goo.gl/BURSaH
33. Ter Louw, M., Venkatakrishnan, V.: Blueprint: Robust prevention of cross-site scripting attacks for existing browsers. In: Security and Privacy, 2009 30th IEEE Symposium on. pp. 331–346. IEEE (2009)
34. Thomas, K., Li, F., Zand, A., Barrett, J., Ranieri, J., Invernizzi, L., Markov, Y., Comanescu, O., Eranti, V., Moscicki, A., Margolis, D., Paxson, V., Bursztein, E.: Data breaches, phishing, or malware?: Understanding the risks of stolen credentials. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. pp. 1421–1434. CCS '17, ACM, New York, NY, USA (2017). https://doi.org/10.1145/3133956.3134067, http://doi.acm.org/10.1145/3133956.3134067
35. Visa Inc: 3D Secure (2017), https://goo.gl/TZSTEc
36. Visa Inc: Visa Developer Centre (2018), https://goo.gl/8dDqWv
37. WickyBay: FRAUDFOX VM, WickyBay Store (2017), https://goo.gl/aAZY1K
38. Zeltser, L.: (2018), https://zeltser.com/malware-sample-sources/

encode_deviceprint()

version%3D2%26pm%5Ffpua%3Dmozilla%2F5%2E0%20%28windows%20nt%2010%2E0%3B%20win64%3B%20x64%29%20applewebkit/537%2E36%20%28khtml%2C%20like%20gecko%29%20chrome62%2E0%2E3202%2E94%20safari/537%2E36%7C5%2E0%20%28Windows%20NT%2010%2E0%3B%20Win64%3B%20x64%29%20AppleWebKit/537%2E36%20%28KHTML%2C%20like%20Gecko%29%20Chrome/62%2E0%2E3202%2E94%20Safari/537%2E36%7CWin32%7Cen%2DUS%26pm%5Ffpsc%3D24%7C1280%7C720%7C680%26pm%5Ffpsw%3D%26pm%5Ffptz%3D5%2E5%26pm%5Ffpln%3Dlang%3Den%2DUS%7Csyslang%3D%7Cuserlang%3D%26pm%5Ffpjv%3D0%26pm%5Ffpco%3D2

asyncpost_deviceprint(url)

dmVyc2lvbiUzRDElMjZwbSU1RmZwdWElMORtb3ppbGxhJTJGNSUyRTAlMjAlMjh3aW5kb3dzJTIwbnQlMjAxMCUyRTAlM0IlMjB3aW42NCUzQiUyMHg2NCUyOSUyMGFwcGxld2Via2l0LzUzNyUyRTM2JTIwJTI4a2h0bWwlMkMlMjBsaWtlJTIwZ2Vja281MjklMjBjaHJvbWUvNjUlMkUwJTJFMzMyNSUyRTE4MSUyMHNhZmFyaS81MzclMkUzNiU3QzU1MkUwJTIwJTI4V2luZG93cyUyME5UJTIwMTAlMkUwJTNCJTIwV2luNjQlM0IlMjB4NjQlMjklMjBBcHBsZVdlYktpdC81MzclMkUzNiUyMCUyOEtIVE1MJTJDJTIwbGlrZSUyMEdlY2tvJTI5JTIwQ2hyb211LzY1JTJFMCUyRTMzMjU1MkUxODE1MjBTYWZhcmkvNTM3JTJFMzY1NONXaW4zMiU3Q2VuJTJEROIlMjZwbSU1RmZwc2MlMOQyNCU3QzEzNjAlNOM3NjglNOM3MjglMjZwbSU1RmZwc3clMOQlMjZwbSU1RmZwdHolMOQxJTI2cGOlNUZmcGxuJTNEbGFuZyUzRGVuJTJEROIlNONzeXNsYW5nJTNEJTdDdXNlcmxhbmclMOQlMjZwbSU1RmZwanYlMOQwJTI2cGOlNUZmcGNvJTNEMQ==

Fig. 5: Device fingerprint information encoded and sent to ACS.

3DS 2.0 Cookies
TESTCOOKIE=Y

ID Cookies
DMC=AiZVNMlzeO1ukqlXqlc7y%2BkM5Vi%2FGf%2Fa1DlCXYyox7%2F
XIr4kfbIlXO4cU%2Bc%2BgWifX5WmJxQFY%2Fl8fH2ysgUzk3FUyhV
jlih3wcIxlG17uFJgBtWgMiZNjoRU6zut3NLLmlXPYLocrIlecsFsRW w%2B6D6JRuya4fb
Hmsww1DOogjzLL4ltobs%3D
cy_track_user=C.28474910.1603347569
3DSSTBIP=yHWvyRz68jCQRAI7zSC3a5YqJJYDrgbtKRs50bDYIkJTU
Xik3MMi6BYEz5zbiXOawTcVFYARXRLY

Fig. 6: Device fingerprint information encoded and sent to ACS.

A Data Used for Transaction Risk Assessment

Table 2 shows an exhaustive list of device attributes from card C1 to C5 that are passed from WB to the ACS. The loading and execution of dfp.js by the ACS as a part of the checkout process is similar for all our test cards that we used. The 'Method' column indicates the functions implemented in the dfp.js that extract information from WB (for readability, in some cases we have simplified the method name). The details that are fetched in each function are shown in 'Attribute description' column of the table. The 'Source' column marks the origin of each attribute (JavaScript or HTTP). Finally, the rightmost column shows an example output value of each function.

Figure 5 and Figure 6 show the encoded devide fingerprint and the full cookie content, respectively.

Table 2: Data used for Transaction Risk Assessment extracted by javascript file dfp.js.

Method	Attribute description	Source	Example values
nav.userAgent()	User agent(UA), OS	JavaScript	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/60.0.3112.113 Safari/537.36
test()	Accepted MIME types/ Documents	HTTP header	text/html, application/xhtml+xml, application/xml;q=0.9, image/webp, image/apng, */*q=0.8
	Accepted Charsets	HTTP header	utf-8, iso-8859-1;q=0.5
	Accepted Encodings	HTTP header	gzip deflate
	Accepted Languages	HTTP header	en-US, en; q=0.8
	ActiveX, GeckoActiveX	HTTP header	?1:0
	Adobe Reader and components	HTTP header	?1:0
	XMLHttpReqest, Serializer, Parser support	HTTP header	Yes/No
deviceprint_ browser()	UA(Version, cpuClass, language)	JavaScript	5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/60.0.3112.113 Safari/537.36; Win32; en-US
	navigator.appName	JavaScript	Netscape
	navigator.appCode Name	JavaScript	Mozilla
	navigator.appVersion	JavaScript	5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/61.0.3163.100 Safari/537.36
	navigator.appMinor Version	JavaScript	5.0
	navigator.vendor	JavaScript	GoogleInc
	navigator.userAgent	JavaScript	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/65.0.3325.181 Safari/537.36
	navigator.oscpu	JavaScript	Windows NT 10.0
	navigator.platform	JavaScript	Win32
	navigator.security Policy	JavaScript	US & CA domestic policy or Export Policy
	navigator.onLine	JavaScript	True
	info_browser.name	JavaScript	Chrome
	info_browser.version	JavaScript	61.0.3163.100
	info_layout.name	JavaScript	Webkit
	info_layout.version	JavaScript	536.36
	info_os.name	JavaScript	win
	navigator.geoLocation	JavaScript	?1:0
deviceprint_ display()	Screen's (colorDepth, width, height, availHeight, availWidth, HDPI, VDPI, Pixel Depth, ColorDepth, bufferDepth, FontSmoothing, Update interval)	JavaScript	2560*1440; 2560*1400; 24; 24
window_ information()	innerWidth, innerHeight, outerWidth, outerHeight, length	JavaScript	675,473,1392,760,3
DoNotTrack	navigator.doNot Track	JavaScript	?1:0
Useofadblock	alert_test	javaScript	?1:0
deviceprint_ software()	Plugins installed	JavaScript	Adobe Acrobat, Macromedia Flash, Java, MS office, Cortana...
deviceprint_ time()	TimeZone	JavaScript	-60
deviceprint_ java()	Java enabled	JavaScript	?1:0
	Java Supported	JavaScript	?1:0
	Java Version	JavaScript	1.6. 1.8
	javaScript cookies support	JavaScript	?1:0
	Server cookies support	JavaScript	?1:0
	HTTP only support	JavaScript	?1:0
flashscript	Flash Version	FlashScript	WIN 28,0,0,126
	Flash Version	JavaScript	28,0,0
	Flash Details	FlashScript	Platform, Major Version, Minor Version, Capabilities (Audio, Accessibility, Audio support, MP3 support, Language, Manufacturer, OS, Pixel aspect, Color support, Dot per inch, Horizontal size, Vertical size, Video
	Number of Fonts	FlashScript	226
	List of Fonts	FlashScript	List of Fonts
deviceprint_ cookie()	Cookie enabled	JavaScript	?1:0
	Session cookie	HTTP header	!yEpKXp9eMDojNcc7zSC3a5YqJJYDrqVB23 H1Cy/yThmhX+omXVM933/...AIr8S7ldvbA==
	Test cookie	HTTP header	TESTCOOKIE=Y
	IDCookie	HTTP header	35BWzcxFkUu1aDdY%2B%2FxvL3VrDuvgoXau%2FAgU %2BJqzYvZZoWiGPKKeYruvsGaPTeecduMcSLa%2FU lf1QGU07S89bddR3dVSFT2dwVeUOd%2FkXvaw7JknH xjFlk4...GY4I7drTK0nT_CNJ%2BhHYW8Y5Wis%3D