Transaction Generators: Root Kits for Web[*]

Collin Jackson
Stanford University

Dan Boneh
Stanford University

John Mitchell
Stanford University

Abstract

Current phishing attacks focus primarily on stealing user credentials such as passwords. In response, web sites are deploying stronger authentication and back-end analytics systems that make it harder for phishers to extract value from stolen passwords. As defenses against traditional phishing improve, we expect to see huge growth in the use of a different type of malware called a *Transaction Generator* (TG). Instead of relying on stolen credentials, a TG simply waits for the user to log in to his account and then issues transactions on behalf of the user. Since strong authentication is ineffective against TGs, mitigation must focus on transaction integrity. We discuss rootkit-like methods that allow TGs to hide their tracks, and explore a number of mitigation techniques, including transaction confirmation. These results suggest that recent identity systems such as CardSpace and OpenID must also address transaction integrity.

1 Introduction

Current phishing attacks steal user credentials, either by directing users to a spoofed web page that fools them into revealing a password, or by installing key-logging malware that records user passwords and sends them to the phisher. In response, web sites are deploying a variety of back-end analytic tools [4, 10, 12] that use past user behavior to determine transaction risk, such as the time of day when the user is typically active and the user's IP address and location. Some sites are moving to stronger authentication using one-time password tokens such as RSA SecurID [14]. These methods, as well as many other anti-phishing proposals [13, 6, 9, 7, 15, 5], focus primarily on reducing the value that phishers derive from stolen passwords.

Fortunately for thieves, and unfortunately for the rest of us, a new form of attack using a *Transaction Generator* (TG) allows criminals to manipulate user accounts directly without stealing user credentials or subverting authentication mechanisms. TG attacks generate fraudulent transactions from the user's computer, through malicious browser extensions, after the user has authenticated to the site. A TG quietly sits on the user's machine and waits for the user to log in to a banking or retail site. Once the authentication completes, web sites typically issue a session cookie used to authenticate subsequent messages from the browser. These session cookies reside in the application environment and are fully accessible to malware. A TG can thus wait for the user to securely login to the site and then use the session cookie to issue transactions on behalf of the user, transferring funds out of the user's account or purchasing goods and mailing them off as "gifts". To the web site, a transaction issued by a TG looks identical to a legitimate transaction issued by the user — it originates from the user's normal IP address at the usual time of day — making it hard for analytic tools to detect. Since TGs typically live inside the user's browser as a browser extension, SSL provides no defense against a TG. Moreover, a clever TG can hide its transactions using stealth techniques discussed in the next section. To date we found only few reports of TGs in the wild [1], but we anticipate seeing many more reports as adoption of stronger authentication becomes widespread.

In Section 3 we explore a number of mitigation techniques, including transaction confirmation. A transaction confirmation system consists of isolated client-side software and a trusted path to the user that enables web sites to request confirmation for transactions that the site deems risky. We discuss the design of a web-based confirmation system and emphasize that a confirmation component is necessary in identification systems such as CardSpace and OpenID.

At a first glance, a Tranasaction Generator may appear to be related to Cross Site Request Forgeries [3] (CSRF). A CSRF attack is due to an incorrect implementation of user authentication at the web site. To prevent CSRF attacks the web site need only implement a small change to its user authentication system. The modification is transparent to the user. In contrast, transaction generators running inside client browsers are much harder to block. All

[*]Supported by NSF through the PORTIA and TRUST projects.

the proposed defenses in Section 3 require changes to the user experience at the site.

2 Building a Transaction Generator

TGs can lead to many types of illegal activity such as,

- Pump-and-dump stock schemes [11]: the TG buys pre-specified stock on a pre-specified date to artificially increase the value of penny stock.
- Purchasing goods: the TG can purchase goods and have them shipped to a forwarding address acquired earlier by the phisher.
- Election system fraud: for voting-at-home systems, such as those used for collecting share holder votes, a TG can be used to alter votes in one way or another.
- Financial theft: a TG can use bill-pay to transfer funds out of a victim account.

An example. Building a TG is trivial, as shown in the hypothetical example in Figure 1. This Firefox extension waits for the user to land on the `www.retailer.com/loggedin` page, which is reached once the user has properly logged in at the retailer. The TG then issues a purchase request to `www.retailer.com/buy` an orders ten blenders to be sent to some address in Kansas. Presumably the phisher hired the person in Kansas to ship the blenders to an offshore address. The person in Kansas (a.k.a mule) may have no idea that he is involved in illegal activity.

2.1 Stealthy Transaction Generators

Transactions generated by a TG will show up on any transaction report page (e.g. an items purchased page) at the web site. A clever TG in the user's browser can intercept report pages and erase their own transactions from the report. As a result, the user cannot tell that fraud occurred just by looking at pages at the site. For example, the single JavaScript line in Figure 2 removes all table rows on a transaction history page that refer to a blender. We have tested this code on several major retailer web sites.

Moreover, suppose the user pays credit card bills online. The TG can wait for the user to log in to her credit card provider site and then erase the fraudulent transactions from the provider's report page, using the same one line of JavaScript shown above. The sum total amount remains unchanged, but the fraudulent transaction is not in the list of transactions. Since most consumers do not bother to check the arithmetic on report pages from their bank, the consumer will pay her credit card bill in full and remain unaware that the bill includes a stealthy fraudulent transaction. This behavior is analogous to how root-kits hide themselves by hiding their footprints on the infected system.

The net result of stealth techniques is that the consumer will never know that her machine issued a non-confirmed transaction and will never know that she paid for the transaction.

3 Countermeasures

We discuss three potential mitigation techniques against the stealthy TGs discussed in the previous section. The first two are easy to deploy, but can be defeated. The third approach is the one we advocate.

1. CAPTCHA. A CAPTCHA on the retailer's checkout page will make it harder for a TG to issue transactions automatically. Retailers, however, balk at this idea since the CAPTCHA complicates the checkout procedure and can reduce conversion rates. There are also security concerns since phishers can hire real people to solve CAPTCHAs. After all, if one can buy a 50 dollar blender for free, it is worth paying 10 cents for someone to manually solve the challenge CAPTCHA. Alternatively, the malware may try to fool the authenticated user into solving the CAPTCHA for a malicious transaction, while the user thinks they are solving the CAPTCHA for some other purpose. Overall, we believe CAPTCHAs cannot defeat a clever TG.

2. Randomized transaction pages. We mentioned earlier that a stealthy TG can remove its transactions from an online credit card bill, thus hiding its tracks. Credit card providers can make this a little more difficult by presenting the bill as an image or by randomizing the structure of the bill. As a result, it is more difficult for a TG to make surgical changes to the bill.

3. Transaction confirmation — a robust defense. An online merchant can protect itself from TGs by using a confirmation system enabling users to confirm every transaction. The confirmation system should be unobtrusive and easy to use.

Here we propose a simple web-based confirmation system that can be deployed with minimal changes to the web site. The system combines confirmation with the checkout process. On the client-side the system consists of two components:

- A confirmation agent that is isolated from malware infecting the browser. For example, the browser might run in a Virtual Machine (VM) while the agent runs outside the VM. Alternatively, the confirmation agent might live on separate hardware device such as a USB token or a Bluetooth cell phone.

- A browser extension, called SpyBlock, that functions as an untrusted relay between the confirmation agent and the remote web site.

```
<?xml version="1.0"?>
<overlay xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul">
<script>
document.getElementById("appcontent").addEventListener("load", function() {
  var currentLocation = getBrowser().selectedBrowser.contentDocument.location;
  if(currentLocation.href.indexOf("www.retailer.com/loggedin") > 0)
  {
    var xhr = new XMLHttpRequest();
    xhr.open("POST", "https://www.retailer.com/buy");
    xhr.send("item=blender&amp;quantity=10&amp;address=Kansas");
  }
}, true);
</script>  </overlay>
```

Figure 1: A Firefox transaction generator: purchases blender once logged-in page is visited

```
document.body.innerHTML =
     document.body.innerHTML.replace(/<tr>.*?blender.*?<\/tr>/gi,"");
```

Figure 2: JavaScript to remove blender line from transaction history page

```
    if (window.spyblock) {
      spyblock.confirm(document.form1.transaction, {
        observe: function(subject, topic, data) {
          document.form1.transactionMAC.value = data;
    } }; }
```

Figure 3: Confirmation JavaScript on checkout page

We briefly describe the confirmation process. The confirmation agent and remote web site share an ephemeral secret key generated by an identity system such as CardSpace during user login. During checkout the remote web site can request transaction confirmation by embedding the simple JavaScript shown in Figure 3 on the checkout page. This script interacts with the untrusted SpyBlock browser extension that relays the transaction details to the confirmation agent. The confirmation agent displays the details to the user and asks the user to confirm the transaction. If the user confirms, the agent sends back a MAC of the transaction details to the browser extension which then forwards the MAC to the remote web site. The web site verifies that the MAC is valid, and if so, fulfills the transaction.

Security relies on two properties. First, the agent's secret key must be isolated from malware. Second, the confirmation dialog must not be obscured by a malware popup to ensure that the user confirms the correct transaction details. Similarly, malware must be prevented from injecting mouse clicks into the agent's dialog. We discuss our prototype implementation in the next section. Note that simply spoofing the confirmation dialog is of no use to the TG since it cannot generate the necessary MAC itself.

A non solution. Clearly, a potential solution to the TG problem is to prevent malware from getting into the browser in the first place. However, the widespread penetration of end-user machines by spyware and bot networks [8] underscores the vulnerability of many of today's machines to malware attacks. We do not expect this to change any time soon.

4 Implementation

Our prototype confirmation system is built on top of the CardSpace identity system in Microsoft's Vista. We could have used other anti-phishing proposals, based on passwords [13, 6, 7, 15] or security tokens [14, 9]. CardSpace, however, includes a convenient authentication UI. Although CardSpace has a "private" desktop designed for ordinary malware resistance, this desktop may be vulnerable to privileged malware or operating system flaws. We further isolate CardSpace from malware using VMware. CardSpace runs on the host OS while the browser runs in a guest.

The prototype is implemented as a browser extension for Mozilla Firefox. The confirmation agent is implemented as a helper application for CardSpace and runs on the host OS. The agent and the Firefox extension in the guest communicate via network sockets. The agent interacts with the user and with CardSpace to generate the transaction confirmation MACs.

5 Conclusion

Transaction generators are a source of concern for enterprises engaged in online commerce [2]. As stronger authentication systems are deployed, we expect transaction generators to pose an increasing threat. This emerging form of malware hijacks legitimate sessions and generates fraudulent transactions using legitimate credentials, instead of stealing authentication credentials. By operating within the browser, transaction generators can potentially hide their effects by altering the user's view of information provided by any site. Consequently, it is necessary to extend identity systems to include a Transaction Confirmation component. As an example defense, we designed SpyBlock, a browser extension and confirmation agent that provide a simple mechanism for web sites to request confirmation. Our prototype is available at `www.getspyblock.com`.

References

[1] New trojans plunder bank accounts, 2006. `http://news.com.com/New+Trojans+plunder+bank+accounts/2100-7349_3-6041173.html`.

[2] COUNCIL, D.-S. I. T. T., AND THE ANTI-PHISHING WORKING GROUP. The crimeware landscape: Malware, phishing, identity theft and beyond. `http://www.antiphishing.org/reports/APWG_CrimewareReport.pdf`.

[3] Cross site request forgery (CSRF). `http://en.wikipedia.org/wiki/Cross-site_request_forgery`.

[4] Cyota. `http://www.rsa.com/node.aspx?id=3017`.

[5] DHAMIJA, R., AND TYGAR, J. The battle against phishing: Dynamic security skins. In *SOUPS '05: Proceedings of the Symposium on Usable Privacy and Security* (2005).

[6] HALDERMAN, J. A., WATERS, B., AND FELTEN, E. A convenient method for securely managing passwords. Proceedings of the 14th International World Wide Web Conference (WWW 2005), 2005.

[7] JUNG, E. Passwordmaker. `http://passwordmaker.mozdev.org`.

[8] MOSHCHUK, A., BRAGIN, T., GRIBBLE, S., AND LEVY, H. A crawler-based study of spyware on the web. In *Proceedings of the 13th Annual Network and Distributed System Security Symposium (NDSS 2006)* (February 2006).

[9] PARNO, B., KUO, C., AND PERRIG, A. Authentication and fraud detection: Phoolproof phishing prevention. In *Proceedings of Financial Cryptography and Data Security (FC '06)* (2006).

[10] Passmark. `http://www.passmarksecurity.com`.

[11] Pump and dump. `http://en.wikipedia.org/wiki/Pump_and_dump`.

[12] Quova. `http://www.quova.com`.

[13] ROSS, B., JACKSON, C., MIYAKE, N., BONEH, D., AND MITCHELL, J. Stronger password authentication using browser extensions. In *Proceedings of the 14th Usenix Security Symposium* (2005).

[14] RSA SecurID. `http://www.rsa.com/node.aspx?id=1156`.

[15] YEE, K.-P., AND SITAKER, K. Passpet: Convenient password management and phishing protection. In *Proceedings of the Symposium on Usable Privacy and Security (SOUPS)* (2006).