

Detecting Fraud on Websites

According to the police report, Ashley McDowell told deputies that two men in a McDonald's parking lot, one with a gold tooth, approached her and asked her to purchase an iPad (www.thesmokinggun.com/file/wooden-ipad?page=0).

tials or session identifiers used to authenticate the user. Given the small scope, there are many things I'm purposely going to overlook. And, frankly, even with this small of a scope, I'm going to skip numerous topics. Someone needs to write a book on this (see *Detecting Malice*; www.detectmalice.com).

ROBERT FLY
salesforce.com

They said they had acquired iPads in bulk and were selling them for \$300 each. McDowell, having only \$180, negotiated her way to a brand-new iPad for a cut-rate price. Upon transfer of the money, the men drove off in their white Impala with no rims. Great deal.

Unfortunately, when Ashley arrived home and opened the box they gave her, she discovered a block of wood with an Apple logo on it (see Figure 1). Drawn on the block were detailed icons for Safari, mail, photos, and an iPod. Apparently, pressing the Safari icon didn't launch a browser. Not a great deal.

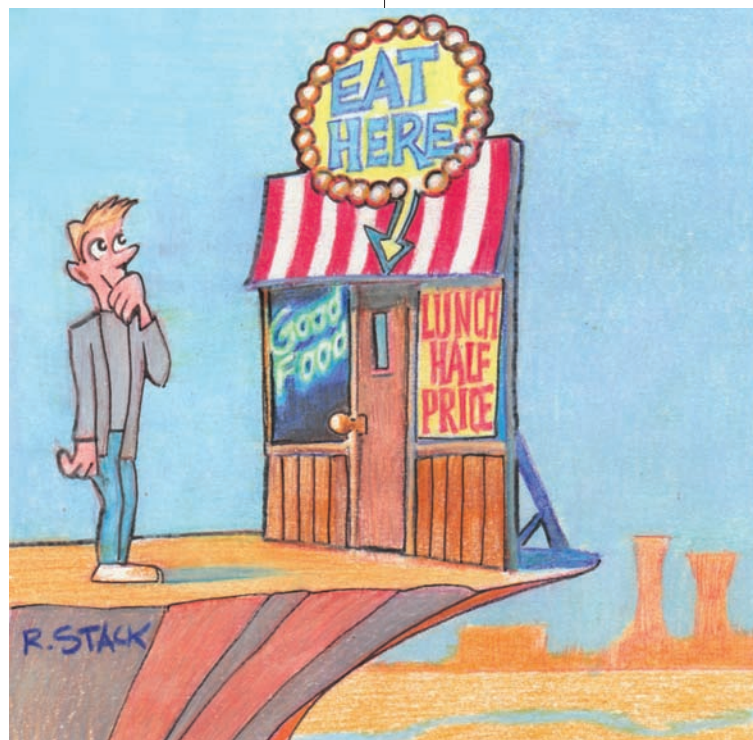
Many people might laugh at this story, thinking that it's absolutely incomprehensible that someone wouldn't bother to see whether he or she was being defrauded. Except it happens all the time.

Yes. Every day. On almost every website. Beyond some specific vertical markets that have been dealing with fraud for ages—such as financial institutions and retailers—most software and services have zero ability to detect someone committing fraud against them or their users. Most sites happily take whatever requests and input users give them, thinking it's legitimate.

Fortunately, there are fairly straightforward things every website can do to detect online fraud. Because fraud can have many meanings to many different people, I want to scope down my meaning of it so that it's accessible by most developers and operations teams. Let's scope it down to this: a nefarious individual who has acquired stolen creden-

Prerequisites

Before jumping into ways to detect potential fraud in Web applications and services, we need to set a few ground rules. Number one, and most important, you *must* have adequate logging. For full coverage on appropriate logging, read "How to Do Application Logging Right."¹ At a minimum, consider the five



types of logging events covered in that article:

- authentication, authorization, and access events;
- changes to the system, application, or data;
- availability issues;
- resource issues; and
- threats or attacks.

Once you have that level of logging, consider whether and how you would expose this information to your customers (assuming you're running a service). It's great that your service has this fantastic logging, but if you have enterprise customers, this level of information regarding their use of your application is just as important to them. Find a way either through your UI or preferably an API to make this information available to them so that they can consume it with their own tools.

Finally, have a third party actively monitor for compromised credentials that show up on nefarious sites and lists. You'll give that party either a domain to monitor for within email addresses (if the service is internal) or one or more login URLs for your service. This way, if the third party detects either of these in the data it finds on compromised servers, you'll have a starting point for finding fraudulent activity. Anyone who has done forensics knows that a single hint regarding a malicious IP address or compromised user is a tremendous starting point for finding many potential instances of fraud.

Getting these fairly simple things going will be a huge boon for some of the more nuanced fraud detection tactics I cover later in this article.

The Golden Rule

The golden rule when building a fraud detection system is to assume that all clients connecting to you are potentially compro-



Figure 1. Sometimes, a \$180 iPad isn't such a great deal after all. (Source: Spartanburg Sheriff's Office; used with permission.)

mised or malicious. Depending on whom you believe, the number of compromised PCs in a given country might sit just around 60 percent.² A chunk of that 60 percent isn't going to be malicious software that hijacks online credentials. However, it's probably also worth pointing out that most antivirus products have, depending on whom you ask, gaps³ or significant gaps⁴ in detection rates. Given that, it's certainly not a far-fetched stance to consider all requests as coming from compromised computers.

Never Trust the Client

Everyone in security knows that servers can't trust clients. In the most basic sense on the Web, this means any sort of client-side validation is useless unless paired with server-side validation.

As a simple example, the following input field allows only two characters:

```
<input type="text"
name="state" id="state"
size="2" maxlength="2" />
```

Under normal circumstances, a user accessing the website through his or her browser would never send a value for the state parameter larger than two characters. In fact, it wouldn't be allowed. A malicious client can easily bypass

this protection, though. Knowing this gives insight to the provider that the user is utilizing the service outside of the normal interface and that subsequent requests during that user's session should be monitored more closely.

Little things like this help build the beginning of a fraud detection engine. A common scenario is to use information such as this and the examples that follow to build a scoring system. Some activity might indicate outright fraud, but most suspicious activity is just that—suspicious. When clusters of several fraudulent indicators appear together for an individual user or your system as a whole, that's a good indicator of misuse.

Impossible Travel

The fastest jet so far has been the SR-71 Blackbird. Its maximum speed is still up for debate, but many sources suggest around 2,200 mph. That means it could travel from California to Romania in a little less than three hours. Of course, there have been no chartered SR-71 passenger flights, but for the sake of argument, imagine that they were possible.

Let's say you saw the two entries shown in Figure 2 in your logs. These entries show that someone accessed login.jsp twice (let's assume the same user and

```
24.4.106.111 - - [28/Jul/2011:10:27:10 -0300] "GET /login.jsp HTTP/1.1" 200 3395
62.217.192.23 - - [28/Jul/2011:12:22:04 -0300] "GET /login.jsp HTTP/1.1" 200 2216
```

Figure 2. Typical lines found in a log file.

```
24.4.106.111 - - [28/Jul/2011:10:27:10 -0300] "GET /login.jsp HTTP/1.1" 200 3395
24.4.106.111 - - [28/Jul/2011:10:27:14 -0300] "GET /home.jsp HTTP/1.1" 200 3148
24.4.106.111 - - [28/Jul/2011:10:27:19 -0300] "GET /account.jsp HTTP/1.1" 200 2349
24.4.106.111 - - [28/Jul/2011:10:27:27 -0300] "POST /transfer.jsp HTTP/1.1" 200 3395
```

Figure 3. Additional log lines showing access to several pages in a fictional website.

```
62.217.192.23 - - [28/Jul/2011:11:22:04 -0300] "GET /login.jsp HTTP/1.1" 200 2216
62.217.192.23 - - [28/Jul/2011:11:22:05 -0300] "POST /transfer.jsp HTTP/1.1" 200 2832
```

Figure 4. Log lines indicating potential fraudulent activity.

that you have those details in the logs) in two hours. At 10 a.m., an IP address in California accessed it; at noon, someone in Romania accessed it. The quick math says that no one could travel from California to Romania in two hours, even in an SR-71.

All that said, this is simply an indicator of potential fraud. Each service and its users are unique; this and other fraud detection techniques aren't definitive indicators of fraud. Depending on the organization, the previous activity might be common. It might be a Romanian businessman who works for a US corporation. Sometimes he connects from his home; other times he connects from his headquarters' virtual private network.

Impossible Travel (Part Deux)

Unlike the SR-71, humans are slow. The SR-71 raced through the air at speeds over 2,000 mph, whereas Usain Bolt (fastest human ever) tops out under 30 mph. Along the same lines, what would the log lines in Figure 3 tell us?

In this example, you see the login page is requested, which then brings up the homepage. Then, the account page is clicked on, and the transfer page is executed. All this happens in just under 20 seconds. That's how people use the Internet. They click links, pages take time to load, and then they click more links after a couple of seconds when they realize what they need to do.

Alternatively, look at the set of requests in Figure 4. First, the request is from Romania again. Depending on your service and users (or a particular user), this might or might not be a concern. More interesting, though, are the progression of URLs and the lack of certain requests.

As I mentioned earlier, humans are slow. We wait for pages to load fully, we like to read a webpage's content when it's being rendered, and some of us might check the URL to make sure the domain we're on is legitimate. How long a person spends on a particular page on a particular website depends 100 percent on that domain, but one thing is for sure, we're not

fast. In the previous set of requests, we see a request for the login page immediately followed by one for `transfer.jsp`. How long did it take? One second. Humans are almost never that fast, especially when you add the network latency and server response time, which, depending on the connection and site, is likely several hundred milliseconds (at least) on its own.

The other problem you'll quickly notice is the jump from `login.jsp` to `transfer.jsp`. This sequence skipped the standard browser progression of pages (read as mouse clicks the user made). Although in some cases users can set bookmarks to particular common actions and resources, there are many actions in a site for which they either wouldn't or couldn't do this. You can track sensitive pages or actions to determine whether the site's use, from the user's perspective, is outside normal standard user behavior.

Knowing what we know about a particular user's use of the site, and the standard use of the site in general, the previous case pretty clearly indicates fraud.

```

POST / HTTP/1.1
Host: login.salesforce.com
Connection: keep-alive
Referer: https://login.salesforce.com/
Cache-Control: max-age=0
Origin: https://login.salesforce.com
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_5_8) AppleWebKit/535.1 (KHTML, like
  Gecko) Chrome/13.0.782.218 Safari/535.1
Content-Type: application/x-www-form-urlencoded
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Encoding: gzip,deflate,sdch
Accept-Language: en-US,en;q=0.8
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.3
Cookie: s_vi=[CS]v1|25E25155851D191C-600001338014E94F[CE]; __qca
  =P0-1224150713-1273812077056; VISITORID=1685926032; sfdc_lvr2=KxG83HMHTCKFzUMIQA
  YWofL5GMb1/PaA8YjF1/f8pI8tDtyKcqSIFFUy39UrNLsZ7mMdNzbEcmPsdWmaiCHAdkmatd9LJOMW/
  BJhd7AplJhNYcdbWgRnttLSboYZRCwoSd90kQGB2iNw==; s_cc=true; rememberUn=false; com.
  salesforce.LocaleInfo=us;

un=ieeesecurity%40gmail.com&width=1440&height=900&username=ieeesecurity%40gmail.com&pw=Stron
  gP%40ssword1&Login=Login

```

Figure 5. A normal HTTP request. This request adheres to the browser's standard format.

The Anatomy of an HTTP Request

As a colleague of mine likes to say, “HTTP requests are not like individual snowflakes.” If you look at them closely, you’ll notice many patterns. Many are browser specific; even more are shared patterns across browsers due to request for comment (RFC) requirements. If you’re really inclined after reading this article, check out the HTTP/1.1 RFC (www.w3.org/Protocols/rfc2616/rfc2616.html) and see how browsers treat the “MUST,” “SHALL,” “SHOULD,” “RECOMMENDED,” and “OPTIONAL” items. You’ll find that different browsers have nuances that will come in handy. This advice will make much more sense as you continue to read.

Let’s look at a standard HTTP request (some portions removed for space). In this case, I’m logging into `salesforce.com` using Chrome (see Figure 5).

We can clearly see that POST is

capitalized, each header capitalizes the first letter of each word, and spacing is uniform, among other things. If I were to send another request from Chrome to the same resource, little, if any, of that uniformity would change.

Attackers, on the other hand, often throw together scripts that don’t mimic the browser exactly. Sometimes headers are missing, capitalization is odd, words are misspelled, and so on. It’s not that far off from years ago when phishing emails were easy to spot owing to formatting and translation issues. Understanding what a real request and an attacker’s request look like comes in handy when you’re trying to find fraud.

Let’s take this a step further. Figure 6 is another request to `login.salesforce.com`, using Firefox from the same machine. If we compare the two requests closely, we see numerous similarities. These consistencies can help us determine whether it’s definitely not a browser accessing the service.

Just as interesting are the inconsistencies in the requests. Looking beyond the `User-Agent` header, what has changed?

- The order of headers differs.
- The order of cookies differs.
- The headers differ. Only the first request has `Origin` and `Cache-Control`; only the second request has `Keep-Alive`.
- Regarding `Accept-Encoding`, only the first request lists `sdch`.
- Regarding `Accept-Language`, the first request has `en-US` and `q=0.8`, whereas the second request has `en-us` and `q=0.5`.

And so on.

Browsers, like any software, follow patterns. They don’t decide to randomly capitalize words or add more spacing. They’re fairly static, and even browser upgrades tend to exhibit many of the same behaviors from release to release.

With this knowledge of how different browsers present requests to servers, you can build

```
POST / HTTP/1.1
Host: login.salesforce.com
User-Agent: Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10.5; en-US; rv:1.9.2.18)
  Gecko/20110614 Firefox/3.6.18
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 115
Connection: keep-alive
Referer: https://login.salesforce.com/
Cookie: s_vi=[CS]v1|26B1AA6785148976-60000166C0016A95[CE]; web_core_geoCountry=us;
  rememberUn=false; com.salesforce.LocaleInfo=us; oinfo=c3RhdHVzPUZSRUUmHlwZT0zJm9pZD0wMEQ4
  MDAwMDAwMEt5TTU=; autocomplete=0
Content-Type: application/x-www-form-urlencoded

un=ieeesecurity%40gmail.com&width=1440&height=900&username=ieeesecurity%40gmail.com&pw=Stron
gP%40ssword1&Login>Login
```

Figure 6. An HTTP request from Firefox. This request varies from other browsers' standard format; for example, the order of headers and cookies differs. The variations form a basis for detecting potential fraud.

detection tools that help identify fraud simply on the basis of whether a browser's requests follow the patterns it exhibits in real use. Analyzing different strands of malware, you'll find that the headers in an HTTP request will routinely appear in a different order and that their values will use distinctly different formats. The exception to this is man-in-the-browser attacks. These are much more difficult to detect and will likely proliferate for years to come because they make fraud detection more difficult, although not impossible.

Fraud Detection in Sessions

When a user logs in to a website, he or she receives a unique session identifier that's good usually for anywhere from a few minutes to a few hours. The vast majority of the time, the session identifier is stored in a session cookie. Such a cookie's name and value pair might look like this: `jsessionid=0a304cd0a304b2549000a30456c5232802`.

Browsers don't share cook-

ies. This is a good thing and helps you out immensely. If your application ever sets a unique, user-specific token saved as a cookie (such as a session identifier), track what browser it was issued to. If a different browser accesses your service using the same session identifier, it's definitely fraud because Firefox doesn't share cookies with Chrome, which doesn't share with Internet Explorer, and so on.

Another highly recommended tactic is to chew your cookies in session. Malware or cross-site-scripting attacks that steal cookies and ship them to a malicious website might not use them for several minutes or hours. Replacing your session identifiers every so often not only helps reduce that risk but also is a giveaway of fraudulent activity because the browser should never replay cookies you've replaced.

System Attacks

Most of the detection techniques I've covered have been user-level detection, where a bad guy is targeting a particular user. You

can do several things to look for broader-based attacks. One of these might occur if several of your users are compromised by similar malware strains and the attacker writes a script to automate the actions on your site.

First, look for lazy attackers. Although attackers usually have access to armies of bots, they sometimes rely on a few IP addresses to pull off their dirty work. Look for places where one IP address accesses multiple users (or companies, depending on how your multitenancy is broken down) and performs the same sensitive actions. Obviously, in some cases this could be legitimate (for example, an organization uses an API to update multiple instances of information or configuration data). However, cases definitely occur in which it isn't, particularly when the information is pulled via standard browser-based HTTP requests (read as not through an API).

Second, use honeypots to your advantage. In your application, create data that's known to catch fraudsters. You can do this

in many ways. In particular, great ways to find out whether attackers have accessed or are trying to access information they shouldn't be privy to when using the site legitimately include

- hidden email addresses,
- bogus user accounts (in your system or fed to malicious sites), and
- known honeypot URLs or parameters.

Whenever you see this information accessed or used, you'll have a great starting point for investigating potential fraud.

Now What?

Assuming you've implemented the techniques I've discussed, you're bound to find some potentially fraudulent patterns. Forensics on a service is a topic deserving its own article, so I'll assume you've got a handle on that and followed the trail of similar IP addresses and application usage that might lead you to additional fraud.

From a service standpoint, though, it's important that you have the following mechanisms in place.

First, you need to alert the customer. This sounds simple, but often your point of contact with the customer isn't the right individual to alert. It's one thing to alert the end user at an organization that fraud occurred; it's quite another thing to tell the user that corporate data was leaked. You must also contact IT, security, or another significant individual when conveying this information. A better approach is to ask for a security contact when the organization first signs up for the service.

Second, you'll need functionality to lock the account as necessary in either real time or close to it. The last thing you want is further access by the malicious party. Lock the account only when you're almost certain of fraudulent activ-

ity; locking users out when you're unsure doesn't necessarily go over that well when you're wrong. Additionally, require that users reset their password once their machine has the malware cleaned up, if necessary. You would be surprised at the number of compromises that occur after users have already been informed that their machine has been infected.

Finally, if the fraud is still occurring, you'll need a method to contain it. This can be as simple as a mechanism in your service that lets you blacklist certain URLs or a regular expression of parameters. This isn't perfect but can come in handy when you must respond quickly. Even more useful is implementing step-up authentication techniques. Out-of-band one-time tokens for sensitive actions, particularly when known fraud is happening, can stop most attacks.

As you've seen, there are many different ways to detect application fraud. If you have the engineering resources, start by knocking off one or two of these and see how that works as you grow the tools. If these resources are at a premium, several commercial products are available. However, depending on your application and its scale, they might not fit your needs and will likely require substantial configuration. Also keep an eye on the Open Web Application Security Project's AppSensor Project (www.owasp.org/index.php/OWASP_AppSensor_Project); it's a free, open source attack detection tool. As it matures, it might prove useful for detecting fraud.

I know I haven't covered many fraud detection techniques—for example, profiling a user's use of the site over time and keeping track of known bad IP addresses accessing your service. However, I hope this has been a good introduction

to some old and new avenues for building tools to help ensure the trust of your user base. □

References

1. A. Chuvakin and G. Peterson, "How to Do Application Logging Right," *IEEE Security & Privacy*, vol. 8, no. 4, 2010, pp. 82–85; doi: 10.1109/MSP.2010.127.
2. "PandaLabs Quarterly Report: July–September 2011," Panda Security, 3 Nov. 2011; <http://press.pandasecurity.com/wp-content/uploads/2011/10/PandaLabs-Report-Q3-2011.pdf>.
3. "On-Demand Detection of Malicious Software," AV-Comparatives, 2010; www.av-comparatives.org/images/stories/test/ondret/avc_report25.pdf.
4. D. Goodin, "Anti-virus Detection Gets Worse," *Channel Register*, 21 Dec. 2007; www.channelregister.co.uk/2007/12/21/dwindling_antivirus_protection.

Robert Fly is the Vice President of Product Security at *salesforce.com*. Contact him at rffly@salesforce.com.

cn Selected CS articles and columns are also available for free at <http://ComputingNow.computer.org>.



The magazine of computational tools and methods.

MEMBERS \$49	CiSE addresses large computational problems by sharing
STUDENTS \$25	<ul style="list-style-type: none"> • efficient algorithms • system software • computer architecture

www.computer.org/cise
<http://cise.aip.org>

This article was featured in

computing **now**

ACCESS | DISCOVER | ENGAGE

For access to more content from the IEEE Computer Society,
see computingnow.computer.org.



IEEE

IEEE  **computer society**

Top articles, podcasts, and more.



computingnow.computer.org