The Goals and Challenges of Click Fraud Penetration Testing Systems

Carmelo Kintana*, David Turner*, Jia-Yu Pan*, Ahmed Metwally*, Neil Daswani*, Erika Chin§†, Andrew Bortz§‡, and The Google Ad Traffic Quality Team*
*Google, Inc.
†University of California, Berkeley
‡Stanford University

***Abstract*—It is important for search and pay-per-click engines to penetration test their click fraud detection systems, in order to find potential vulnerabilities and correct them before fraudsters can exploit them. In this paper, we describe: (1) some goals and desirable qualities of a click fraud penetration testing system, based on our experience, and (2) our experiences with the challenges of building and using a click fraud penetration testing system called Camelot that has been in use at Google.**

I. Introduction

Click fraud is the act of clicking ads with fraudulent or malicious intent to generate illegitimate revenue or hurt competitors. Since online advertising networks act as brokers of multi-billion dollar online advertising revenue streams, click fraud is a major responsibility and concern. While Google needs to be cautious with exposing information about its click fraud detection techniques to preserve its effectiveness, [1] reports on reasonable countermeasures that Google deployed, and in [2], [3], Daswani *et al.* gives a panoramic treatment of the types of click fraud, countermeasures, and detection filters that Google employs.

Click fraud filters mark a click as *invalid* to not charge the advertiser for malicious or poor quality clicks [2]. Determining whether a particular click is correctly marked as invalid by a set of filters is fundamentally a difficult problem, since judging a click as fraudulent depends on the clicker's intent, and it is sometimes impossible to definitively ascertain intent. One approach that can be used to measure the false negatives of a set of filters (i.e., the unfiltered malicious clicks) is to inject artificial clicks into a click stream, run the click stream through click filters, and examine how different the set of clicks marked invalid is from the set of artificial clicks.

This paper describes the desirable properties of a practical click fraud penetration testing (hence referred to as *pen-testing*) system and highlights the challenges of realizing such a system. We have built Camelot, a system that has been used by the Google Ad Traffic Quality Team to conduct *isolated* click fraud experiments against our online detection filters to identify potential vulnerabilities under semi-realistic conditions without impacting advertisers and publishers. During pen-testing exercises, Camelot is used to simulate novel attacks, proactively find potential vulnerabilities before malicious external attackers, and correct them when necessary.

Besides conducting penetration tests on click filters, a pen-testing system can be used for other purposes, such as a regression testing tool during the development of new filters. In addition, the pen-testing system may be provided to elect advertisers and academic researchers to assess the effectiveness of the click fraud filters without revealing their implementation details.

We restrict the construction of the pen-testing system to a subset of click filters. Some of the filters at Google are not suitable for pen-testing, because they take input from offline data sources. Moreover, the click filters are just one component of all possible defenses against click fraud; other components include various offline automated detection tools, as well as human reviewers.

The organization of this paper is as follows. Section II describes desirable requirements of a click fraud pen-testing system and Google's implementation of such a system, Camelot. Section III focuses on the challenges of building and using click fraud pen-testing systems, including challenges specific to Camelot. Section IV describes our experience with Camelot. Section V discusses the related work, and we conclude in Section VI.

II. The Goals and Properties of Click Fraud Penetration Testing Systems

A click fraud pen-testing system provides an environment for testers to conduct experimental attacks on a click fraud detection system, with the main goal of *proactively finding vulnerabilities in the system before being exploited by malicious attackers*. By finding security vulnerabilities earlier, the click fraud filters can be improved and malicious attackers have fewer chances of conducting successful attacks. Therefore, a desirable pen-testing system should include several functionalities. In particular, the system should be flexible in emulating attacks at different levels of sophistication (*flexibility*) and be easy to use in conducting experiments (*simplicity*). In addition, for practical reasons, a desirable pen-testing system should have no impact on

§ This work was done while authors were interning at Google, Inc.

real world entities and traffic, and allow experiments to be conducted in an isolated fashion (*isolation*).

Emulating an attack scenario includes setting up the entities (e.g., the clicking users, ad links to be clicked on, content publishers, and advertisers) and the traffic (including query and click traffic) among them. A flexible pen-testing system allows a pen-tester to realistically emulate an attack through properly setting up the parameters of the involved entities and traffic, and to conduct controlled experiments by tuning the parameters to find vulnerabilities in the click fraud detection system. Moreover, the pen-testing system should be easy to use, making the entities and traffic (by simulation) available to the pen-testers and provide fast feedback on the successful rate of the experiments to shorten the experiment cycles.

To achieve the flexibility and simplicity of a practical pen-testing system, we identify a list of properties in Table I. We group these properties into two categories: (1) "Real," the properties needed to simulate attacks in a realistic manner, and (2) "Virtual," those specifically designed to facilitate the pen-testing experiments. A *general characteristic* of the properties in Table I is that no property should make it harder for a pen-tester to conduct an experimental attack compared to conducting the same attack in reality, nor makes detection of the bad traffic any easier by the filters. The list is not intended to be exhaustive, and each of these properties is explained in detail in the remainder of this section.

For a practical system, the Isolation property (V1) is considered a high priority among the properties listed in Table I. In practice, it is crucial for pen-testing experiments not to interfere with the real-world advertisers and traffic. For example, the experimental clicks of the pen-testing system neither should have an impact on the budget of real-world advertisers, nor should the clicks appear in real-world advertisers' log files. Similarly, injecting experimental clicks in real traffic should neither consume production resources, nor increase latency in processing real traffic. Furthermore, our production online filters (and all components of our ad system) should not consider injected traffic when making decisions.

Since the main goal of the pen-testing system is to find vulnerabilities, virtual properties that make it easier for pen-testers to conduct experiments quickly are also desirable to have. These properties include fast feedback, ability to run simultaneous experiments, and virtualizing some resources such as IPs for instance.

A. Properties for Real World Effects

For a pen-testing system to simulate all possible attacks that can be conducted in the real world, it should have functionalities that simulate activities happening in the real world and should provide pen-testers with information that real attackers can gain. We discuss several real-world properties for a pen-testing system in this section.

R1 Traffic Examinations: In the real world, an attacker may have control on real traffic from one or more machines by owning or compromising them. There are two major benefits to having access to real traffic: (1) it allows attackers to learn actual usage patterns, and (2) it enables the ability to "piggyback" the fraudulent traffic on top of actual, legitimate traffic. Knowledge of the normal traffic patterns enables the attacker to generate fraudulent traffic that mimics normal usage. With access to a single compromised machine, an attacker is given a real user's history and future traffic, use of a real browser, real IP address, etc. With access to many compromised machines, an attacker is additionally given a real distribution of machine and user types. A pen-testing system should provide this traffic pattern information to the pen-testers and should allow mixing of organic traffic with the artificial traffic.

For the types of attacks that an attacker can conduct with a compromised machine, an attacker can execute attacks that depend on user behavior, simultaneously borrowing distributions such as query and timing and outputting cover traffic. Moreover, the attacker could "piggyback" clicks, that is, inject fraudulent clicks when users perform real queries and clicks.

R2 All Defenses: To be most realistic, a pen-tester's attack should face all the defenses that a real attacker would face. Besides online click filters, the defenses of a typical ad network provider usually also include query and conversion filters, offline automated detection tools, and human reviewers. Depending on the scope of the penetration testing, a pen-testing system should provide the defense components to be tested. However, many of these may be difficult or impractical to run against for penetration testing. In this study, we focus on pen-testing a subset of the online click filters.

R3 Real Effects: In the real world, a change in traffic pattern (e.g., receiving fraudulent traffic) usually triggers real-time effects on the ad serving network, as well as long-term effects on the advertisers and the publishers. For instance, on an ad serving network, the additional fraudulent traffic consumes an advertiser's budget in real-time and may trigger the ad serving network to stop serving ads if the budget is depleted. At a larger time frame, the fraudulent traffic may trigger an advertiser to change its bidding strategy or a publisher to change the layout of the content pages. It is desirable for a pen-testing system to simulate these real effects, so that the impact of an experimental attack can be evaluated realistically.

Lacking real effect simulation may bias the outcome of an attack. For example, without simulating the budget effect, the ad serving system will continue to present the same set of ads for the same query (i.e., no *ad cycling*) and limit an attacker to a small set of ads to attack. Attacking a small set of ads creates an unrealistic concentration of fraudulent traffic, making the experimental attack easier to catch than

Table I
PROPERTIES OF A PEN-TESTING SYSTEM

REAL		
Number	Name	Description
R1	Traffic Examinations	"Sees" traffic from a real network connection.
R2	All Defenses	*All* production defenses are tested against.
R3	Real Effects	Real effects on advertisers and users
R4	All Advertisers/Publishers	Has access to all advertisers/publishers.
VIRTUAL		
Number	Name	Description
V1	Isolation	Does not influence real advertisers, publishers, and users.
V2	Resource Emulation	Provide virtual resources to conduct more powerful attacks.
V3	Fast and Accurate Feedback	Fast feedback loop with detailed information.
V4	Virtual Time Compression	Speed up simulation time of attack.
V5	Simultaneous Experiments	Runs multiple experiments simultaneously.
V6	Traffic Composition	Total control of traffic writes.

in the real world. On the other hand, without the budgeting effect, a successful experimental attack potentially can achieve unlimited revenue from a single ad and erroneously inflate the actual impact of the attack in the real world. If the goal is to identify potential vulnerabilities in the filters, it is more important to make sure pen-testing attacks are no easier to detect the realistic ones (e.g., due to lack of ad cycling). The fact that some attacks may have greater success than in the real world is less of an issue.

R4 All Advertisers and Publishers: In the real world, an attacker can target any publisher and click on any ad in any content page. To be able to explore the full design space, a pen-tester must be able to attack any actual advertiser on any publisher site. Limiting a pen-tester to a small set of advertisers and publishers can make an experimental attack easier to catch (due to the unrealistic concentration of traffic as discussed for R3).

B. Properties for Virtual Effects

As mentioned in the beginning of this section, a pen-testing system should make the pen-testing experiments easy and safe to conduct with no impact on the real world traffic and entities (the "simplicity" and "isolation" functionalities). In this section, we discuss these additional "*Virtual*" properties in detail.

V1 Isolation: In practice, a pen-tester's experimental attack should not impact any real advertisers, publishers, or users in any way. Additionally, pen-testing should not impose unnecessary impact on our production system. In particular, any click, query, and conversion made by the pen-tester should not appear in production logs and should not affect production filters. Any lack of isolation may also have financial and legal implications. We made this property the top priority when designing our "Camelot" system (described in Section II-C).

V2 Resource Emulation: Resources that are relevant to a pen-testing experiment include machine IPs (for establishing valid TCP/IP connections) and client side state (e.g., cookies). To simplify a pen-tester's job in conducting arbitrary attacks, it is desirable for a pen-testing system to provide virtual resources and allow the pen-tester to focus on finding vulnerabilities. With this property, a pen-tester can also design attack schemes which may not be as easy to conduct in the real world. For instance, an experiment involving a botnet of 10K machines should be easily conducted in the pen-testing system, without needing to obtain access to 10K physical machines as in the real world.

V3 Fast and Accurate Feedback: Another property that makes iterative experiments simpler is fast and accurate feedback, especially in the form of the number of invalid clicks detected from the filters. Compared to the real world scenario, a pen-testing system can provide more timely feedback to the pen-tester. Moreover, feedback to the pen-testers can be more accurate and detailed (all the way to providing results at the click-by-click level).

In the real world, ad networks like Google have worked very hard to prevent malicious attackers from obtaining feedback pertaining to their attacks. For an attacker who targets Google search ads (e.g., to hurt competitor advertisers), one way for the attacker to obtain feedback is to focus her attack on a particular ad until depleting the budget of the ad (i.e., until the particular ad is not served by Google when given the same search keywords). For an attacker who targets content ads (e.g., to make money directly), the attacker can join the ad network as a publisher (e.g., Google's AdSense for Content program) and conduct attacks on her own content pages. The feedback of the content ad attack can be obtained from the reports that the ad network provides to the publisher (same as the attacker in this case). Comparing the feedback for these two kinds of attacks,

the feedback for search ad attacks is ambiguous (e.g., a large budget may lead to an ad to continue to show, or budget may have been depleted by organic traffic) and only gives the attacker a coarseness of the success of the attack. The feedback for content ad attacks is more detailed, but ad networks can also take steps to dampen the feedback information (for example, by providing aggregated statistics to publishers).

V4 Virtual Time Compression: To allow pen-testers to conduct more iterations of pen-testing to find vulnerabilities, it is necessary for the pen-testing system to speed up the simulation time of an experimental attack. In particular, the simulation time should be shorter than the time frame of an attack in real world. For example, it is impractical for a pen-tester to spend three months to conduct an experimental attack that spans three months. We call this property of pen-testing systems the "*virtual time compression*" property.

V5 Simultaneous Experiments: A practical feature for a pen-testing system is to allow a pen-tester to conduct simultaneous, isolated attacks in order to quickly iterate through different experiments. This is not possible in the real world, since the clicks for two independent attacks on production Google Search are interleaved in the same set of logs, possibly causing interactions during click fraud filtering. Isolation, and the ability to replicate instances of the Camelot system, makes it easy for us to provide this ability.

V6 Traffic Composition: To permit more comprehensive exploration of the design space of attacks, a pen-tester should be allowed complete control over all traffic that reaches the ad network during the experiment, including creation or deletion of legitimate background traffic. For instance, if a pen-tester is simulating a 100-machine botnet, the simulation environment should allow her to inject traffic patterns of other uncompromised machines as well. In addition, having this property gives a pen-tester ability to conduct controlled experiments, by tuning particular traffic and studying the corresponding impact.

This property provides a pen-tester more control than what an attacker has in reality, since a pen-tester can fully control traffic from all sources, while a real attacker can only alter the traffic of machines under her control. This property is closely coupled with the R1 property where artificial traffic can be piggybacked on top of legitimate traffic. However, this does not belong to the "Real" category since an attacker cannot prohibit traffic from uncompromised machines from reaching the ad network.

With the properties listed above, we can compare the environment a real attacker faces with the environment provided by our current implementation of Camelot (*Camelot 1.0* is discussed in Section II-C). Table II summarizes the differences between the two environments, with respect to the properties that we listed above. Section III-B discusses the challenges of supporting the real features that are currently not fully supported in Camelot 1.0 and what is required to support them.

Table II
PROPERTIES OF A REAL ATTACK AND CAMELOT 1.0

Environment	Supported	Not Supported
Real Attack	R1, R2, R3, R4	V1, V2, V3, V4, V5, V6
Camelot 1.0	R4, V1, V2, V3, V5, V6	R1, R2, R3, V4

C. Camelot: Google's Pen-Testing System

In this section, we explain the design and properties of Camelot 1.0, Google's current implementation of an isolated pen-testing system. There are two general strategies we used when we designed Camelot 1.0: (1) Achieving isolation via system replication, and (2) achieving flexibility via a two-step procedure (i.e., a click generation step followed by a filtering step). We discuss these design decisions in more detail in this section.

1) Architecture: Of all the properties for a click fraud pen-testing system, isolation is the main property that motivated the design of Camelot to prevent any unwanted effects on real users, advertisers, and publishers. To achieve isolation, we replicate instances of the ad servers and filtering systems. We also replicate the frontends for requesting Google Search ads or AdSense for Content ads. Thus, the Camelot frontends provide an interface to pen-testers that is the same to real attackers.

As seen in Figure 1, pen-testers can develop clickbots and run them against Camelot's frontends while treating the click fraud detection system as a black box. Camelot logs all clicks on ads and blocks redirects to the advertiser's site to avoid affecting the advertiser's server logs. After running an attack, the ad click logs are saved and run through the click filters. Finally, a report that lists the total number of clicks and invalid clicks is generated. Thus, unlike the real world system which processes clicks in real time, Camelot operates in a *batch* mode with two steps: all clicks are generated first in Camelot and then run through the filters. In addition, Camelot also provides penetration testers access to the following virtual abilities to make pen-testing much easier:

(a) Arbitrary IP addresses can be specified on both the queries and ad clicks.
(b) Integrity checks are disabled on certain data in the HTTP request such as cookies.

2) Properties of Camelot: Camelot provides several advantages that makes pen-testing safer and easier than conducting a real world attack. The main advantage of the Camelot system is that it provides isolation for both real advertisers and publishers from click fraud experiments (because of the use of separate non-production ad servers). As a result, all of the pen-tester's clicks and queries go in

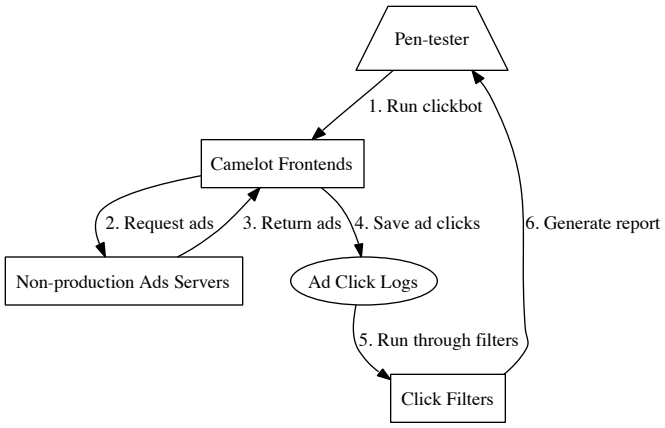

Figure 1. Camelot Pipeline

their own logs and are never logged with real traffic, and thus are never processed by production filters. Furthermore, the use of non-production systems avoids overloading the production systems, further isolating production from click fraud experiments. Because of isolation, any ad may be targeted in an attack.

In addition, pen-testers using Camelot can emulate resources such as arbitrary IP addresses. Some additional advantages of the Camelot system include immediate and accurate feedback as well as the ability to parallelize experiments.

There are a few drawbacks of the current Camelot implementation due to technical limitations. The first disadvantage is that Camelot 1.0 only runs a subset of the online filters, namely the click filters, rather than all possible defenses (e.g., query and conversion filters). Next, clicks generated with Camelot 1.0 are processed by the filters in isolation rather than being merged with real clicks from that same time period. One remaining disadvantage is that it is not possible to actually deplete an advertiser's budget because we did not implement budgeting effects for use by Camelot 1.0's ad servers. Section III-B discusses in more detail how these technical limitations prevent Camelot 1.0 from achieving a number of our desired properties. However, even with these disadvantages, the combination of isolation and virtual abilities makes Camelot 1.0 a useful pen-testing tool.

III. Challenges

In this section, we explore the challenges of pen-testing, including fundamental problems that exist in all pen-testing systems, and problems specific to Camelot. The challenges are followed by discussions on interesting and subtle aspects of click fraud pen-testing.

A. Isolation versus Realism

As we outlined in the previous sections, desired properties of a pen-testing system are sometimes mutually contradictory. Any pen-testing system's design, including Camelot's, must determine which set of tradeoffs to make between these contradictory properties.

In particular, *isolation* of the pen-testing system from real traffic is often in contradiction with the *realism* in interacting with advertisers, publishers, and users. Take for example the ad auction system. For realism, we would like to use the production auction servers so the pen-tester can see real-world ads. However, the experimental clicks that are generated affect the production click-through rate, which violates our isolation invariant. Any injected traffic from experiments on the pen-testing system should not have any effect on real world traffic, users and advertisers.

On the other hand, without any interaction with the real advertisers and publishers, the pen-testing system is less realistic, since it is missing the real-time reactions from the advertisers and publishers. For instance, an advertiser may increase the ad campaign budget, or a publisher may change the ads layout on a site, in response to the additional clicks inserted by the penetration tests. However, since the absence of these effects does not make it more difficult to defeat the filters, the pen-testing system still achieves our goals.

B. Practical Issues of Camelot 1.0

As we explained in the previous section, there are certain features that Camelot 1.0 is missing. In this section, we discuss all of the properties that we do not support along with challenges in addressing these in future systems.

1) R1 Challenges (Traffic Examinations): Clicks are filtered in isolation by Camelot, unlike a real attack where they would be hidden amongst a far larger amount of legitimate cover traffic. The fact that the only clicks being processed by the filters are from the simulated attack gives an unrealistic advantage to the filters.

Since the absence of this property makes the pen-test more difficult than a real attack, this has a high priority among our future directions. Addressing this involves (1) running real logs through Camelot in parallel with the pen-testing

logs, and (2) giving the pen-tester access to the information of each real click.

2) R2 Challenges (All Defenses): Currently, Camelot neither runs the offline detection tools against a simulated attack, nor filters that are based on queries or conversions. Also, there is no human reviewer devoted to manually inspecting suspicious clicks created by the pen-testers. Hence, even if a penetration tester successfully gets clicks marked as valid by the online click filters, there is a chance that these clicks can be detected offline if the attack happens in the real world.

Implementing these additional filters involves adding dozens of distributed processes to the already complex Camelot system. To improve the R2 coverage of Camelot, we plan to target those features that help us find vulnerabilities in the current filters. Enabling query filters is important to our overall goal and will likely be added in the future. Similarly, one of our future directions is adding conversion filters to Camelot. This is not straightforward since conversions are normally generated on the advertiser's website (which contradicts our isolation goal). Therefore, we need a separate conversion simulation system.

3) R3 Challenges (Real Effects): Besides not having the conversion information (as discussed in Section III-B2), another main real effect that is missing from Camelot 1.0 is the effect of ad budget depletion. Roughly, missing this feature causes the same ads to be shown for a particular query regardless of the number of times the search result page has been returned to the pen-tester, or the number of times the ad has been clicked on during the experiment. To our filters, the clicks may look less organic in the pen-test than in a real attack. For example, on a particular query all clicks may concentrate on the same static set of ads. This moves away from our goals, since the lack of this feature makes it harder to defeat the filters than in the real world.

The engineering effort required to simulate a budget server is non-trivial. The system is large and complex, and it relies on and interacts with a number of other systems. In practice, we have not found this to be a large issue in exploring the attack design space, but we may decide to model budgets more realistically in the future.

4) V4 Challenges (Virtual Time): Camelot currently does not support virtual time compression due to some technical difficulties. First of all, this requires existing filters to support the concept of virtual time. A pen-testing scenario that supports the virtual time compression is as follows: a pen-tester is given the ability to provide timestamps on queries and clicks, and the filters and the rest of the ad serving system need to behave according to the virtual time.

An example of a filter that supports virtual time is a simple filter that involves measuring frequency of traffic on a given entity over a period of time. Since the pen-testing system is injecting the traffic at a much faster rate than real time, such a filter needs to also allow for virtual time to be observed. Thus, implementing this property is one of our first future directions since it allows pen-testers to quickly conduct low volume, long term attacks.

IV. Experience with Camelot

We have been using and maintaining Camelot for over a year. One of the main challenges has been maintaining all of the various components of Camelot since we run our own replicated instances of the frontends and ad servers. Most of the time, the frontends properly return ads during a query request, though we've had several occasions where configuration files become broken or the ad servers need to be updated to the latest production version in order to continue serving ads.

The Camelot system has been used internally at Google to conduct penetration tests of the click fraud filters in order to discover vulnerabilities. In addition, new filters can be developed and easily run within Camelot. This allows us to perform regression testing where we can replay old click logs through Camelot in order to ensure that our filters are working as well as they did in the past. Camelot can also be used as part of new filter development to make it a little easier to create an isolated instance and run realistic tests against it.

During the summer of 2008, we also invited graduate students (our co-authors from Stanford and UC Berkeley) to conduct pen-tests using Camelot for research purposes. Experiments were conducted deliberately in a black box manner, to approximate a real-world attack scenario. While every effort was made to avoid disclosing information about click fraud signals or detection systems, it is possible that the pen-testers may have learned some details that they may not have been exposed otherwise. That said, we do not believe that any such details were significant enough to bias or overly impact their approach or the results of their experiments.

Using Camelot, one pen-tester attempted to reverse engineer the click fraud filters, while the other pen-tester attempted a series of experiments aimed at ultimately maximizing the number of clicks not marked as invalid while also minimizing resources (e.g., IPs) used. The results of the reverse-engineering experiment reveal that, while difficult and time-consuming, various characteristics of Google's filters can be identified by a series of carefully designed experiments if the pen-tester is provided accurate feedback. The second set of experiments involved 44 experiments over a period of three months before achieving an attack involving 10K clicks with a low percentage of clicks marked invalid.

Figure 2 shows the percentage of clicks marked invalid per experiment from the second set of experiments. The vertical bars in the chart distinguish five ranges of experiments with similar complexities: experiments 1–8, 9–20, 21–24, 25–36, and 37–44. The first range of experiments is attacks involving at most 10 clicks, while the pen-tester was getting

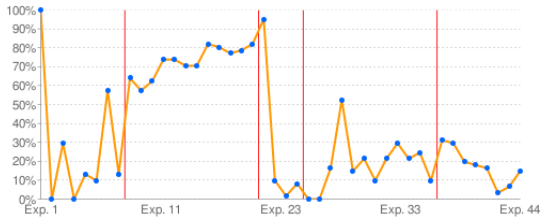

Figure 2. Percentage of clicks marked invalid

familiar with Camelot. The fluctuations of percentage clicks marked invalid are mainly due to the small amount of clicks made in these experiments. The second range shows a gradual increase in percentage of clicks caught as the pen-tester conducted attacks involving at most 100 clicks. The third range comprises attacks involving at most 1K clicks. At this point, the pen-tester was able to refine the clickbot to get most of the clicks through the filters. The final two ranges involved at most 5K and 10K clicks, respectively. During the final experiments, the pen-tester attempted additional refinements to the clickbot and also placed constraints on the number of resources to use which led to the fluctuations in the percentage of clicks caught. As we expected, with enough effort, these results show that scaling an attack to a large and valuable number of clicks is feasible.

These pen-testing experiments would not have been possible without the functionalities provided by Camelot such as the *free* virtual resources (e.g., IPs) which allow for scaling up attacks, and the fast and accurate feedback which speeds up the attack refinement cycle. Since Camelot is not accessible to attackers in the real world, it is virtually impossible for a real attacker to conduct similar test attacks to discover these vulnerabilities. We have developed new filters to catch the vulnerabilities that were exposed by these experiments.

V. Related Work

The research that is related to pen-testing click fraud filters can be categorized into two classes. The first one focuses on pen-testing for intrusion detection. The second class comprises the work done on click fraud detection.

One of the first published works involving pen-testing was in [4], where the Intrusion Detection Systems (IDSs) developed under DARPA funding were evaluated by generating attacks mixed with sanitized live background traffic. However, due to the privacy and technical challenges of obtaining and sanitizing the data, as well as inserting the attacks, the evaluation was done on synthetic traffic that resembles that of the operational network [5]. The option of using realistic, instead of real, background traffic and attacks was criticized in [6]. Among other defects in the evaluation process, McHugh argued the synthetic data could bias the evaluation results, especially since attacks that are unsystematically collected from various sources could be unrepresentative of real-world attacks. The reader is referred to [7] for a more complete survey of intrusion detection pen-testing and benchmarking. The isolation of the pen-testing exercises has been discussed in [8].

The second category of our related work is the academic work that focuses on click fraud detection. While academic researchers have proposed some valuable detection mechanisms for click fraud, the lack of long-term access to live ad serving systems have hindered publishing any tests conducted on these proposals in a large-scale system. To the best of our knowledge, this is the first published work that deals with issues related to pen-testing deployed click fraud filters.

The body of academic research on click fraud is relatively small. The significance of the problem has been studied in [9] that has shown search engines that combat fraud best have a competitive advantage since advertisers would have better Return On Investment (ROI). This would in turn, not only raise the bids on the ads of this broker, but also attract more advertisers.

Meanwhile, the countermeasures have been the focus of most of the academic research [10], [11], [12], [13], [14], [15]. Overall, there have been three main comprehensive approaches to combat click fraud. The first approach [16], [17], [14] addresses the problem radically by proposing tracking surfers using cryptographic primitives, and identifying human generated click from "legitimate" surfers. However, the scheme either violates the surfers' privacy by identifying them or reduces the problem of click fraud to an unsolved problem like credit card fraud.

The second approach [18] focused on how to make the payment system resilient to click fraud by removing its incentive. In [18], Immorlica *et al.* proposed a class of CTR measurement algorithms that converts detecting click fraud to detecting impression fraud. The approach fails to hinder fraudsters from simulating huge traffic with low CTR. Thus, the approach advocates formalizing the problem of detecting fraudulent traffic as an arms race between fraudsters and brokers on how much traffic can fraudsters generate without being detected.

The third approach [19], [13], [20] gives an advantage to brokers in the arms race. The approach depends on the correlation between the success of an attack and the amount of resources at the fraudster's disposal. The approach proposes traffic analysis techniques that identify "hot spots" of traffic between the attacked entities (ads or publishers' sites) and the relatively limited attacking resources (e.g., machines) that are temporarily identified by cookies and IPs. In [19], [13], [20] traffic analysis techniques were proposed whose cost is significantly less than the cost incurred on the fraudsters to circumvent them.

VI. Conclusion

Penetration testing a click fraud detection system requires system designers to think about various factors such as (1) how to simulate realism, (2) how to ensure isolation such that no side-effects can impact real advertisers and publishers, and (3) how to virtualize such that penetration testers can conduct attacks faster and more efficiently than real fraudsters can. In this paper, we have enumerated numerous desirable properties that one might like in a click fraud pen-testing system, and outlined some of the trade-offs involved in supporting such properties in a system called Camelot that has been in use at Google to conduct click fraud penetration tests.

Camelot allows pen-testers to attack Google's click fraud filters in an isolated manner. The Camelot system was used to allow full-time employees as well as interns to conduct black box penetration testing experiments.

The results of these tests provide not only insight into Google's own filters but also provide insight into the iterative attack development process that real fraudsters might employ. The Camelot system has become a practical and useful platform for click fraud penetration testing within Google. We hope and expect that search and per-pay-click engines continue to employ such types of systems to proactively find vulnerabilities to help defend and improve the return on investment that advertisers receive, and the revenues that publishers receive.

Acknowledgments

We thank Thomas Duebendorfer, Kourosh Gharachorloo, Zoltan Gyongyi, Yechiel Kimchi, Matt Paduano, and Razvan Surdulescu for providing helpful feedback.

References

[1] Tuzhilin, A., "The Lanes Gifts v. Google Report," 2006.

[2] N. Daswani, C. Mysen, V. Rao, S. Weis, K. Gharachorloo, and S. Ghosemajumder, "Online Advertising Fraud," *Crimeware: Understanding New Attacks and Defenses*, 2008.

[3] N. Daswani and M. Stoppelman, "The anatomy of Clickbot.A," in *HotBots*, 2007, p. 11.

[4] R. K. Cunningham, R. P. Lippmann, D. J. Fried, S. L. Garfinkel, I. Graf, K. R. Kendal, S. E. Webster, D. Wyschogrod, and M. A. Zissman, "Evaluating Intrusion Detection Systems without Attacking your Friends: The 1998 DARPA Intrusion Detection Evaluation," in *SANS*, 1999.

[5] R. P. Lippmann, J. W. Haines, D. J. Fried, J. Korba, and K. Das, "Analysis and Results of the 1999 DARPA Off-Line Intrusion Detection Evaluation," in *RAID*, 2000, pp. 162–182.

[6] J. McHugh, "Testing Intrusion Detection Systems: a Critique of the 1998 and 1999 DARPA Intrusion Detection System Evaluations as Performed by Lincoln Laboratory," *ACM Trans. Inf. Syst. Secur.*, vol. 3, no. 4, pp. 262–294, 2000.

[7] N. Athanasiades, R. Abler, J. Levine, H. Owen, and G. Riley, "Intrusion Detection Testing and Benchmarking Methodologies," in *IWIA*, 2003, p. 63.

[8] C. Hurley, A. W. Bayles, K. Butler, A. C. John, E. Miller, and G. M. Phillips, *Penetration Tester's Open Source Toolkit*. Syngress, 2007.

[9] B. Mungamuru and S. Weis, "Competition and Fraud in Online Advertising Markets," in *FC*, 2008, pp. 187–191.

[10] V. Anupam, A. Mayer, K. Nissim, B. Pinkas, and M. Reiter, "On the Security of Pay-Per-Click and Other Web Advertising Schemes," in *WWW*, 1999, pp. 1091–1110.

[11] D. Klein, "Defending Against the Wily Surfer-Web-based Attacks and Defenses," in *USENIX ID*, 1999, pp. 81–92.

[12] A. Metwally, D. Agrawal, and A. El Abbadi, "Using Association Rules for Fraud Detection in Web Advertising Networks," in *VLDB*, 2005, pp. 169–180.

[13] A. Metwally, D. Agrawal, A. El Abbadi, and Q. Zheng, "On Hit Inflation Techniques and Detection in Streams of Web Advertising Networks," in *ICDCS*, 2007, p. 52.

[14] M. Naor and B. Pinkas, "Secure and Efficient Metering," in *EUROCRYPT*, 1998, pp. 576–590.

[15] L. Zhang and Y. Guan, "Detecting Click Fraud in Pay-Per-Click Streams of Online Advertising Networks," in *ICDCS*, 2008, pp. 77–84.

[16] C. Blundo and S. Cimato, "SAWM: A Tool for Secure and Authenticated Web Metering," in *SEKE*, 2002, pp. 641–648.

[17] A. Juels, S. Stamm, and M. Jakobsson, "Combating Click Fraud via Premium Clicks," in *USENIX Security Symposium*, 2007, pp. 17–26.

[18] N. Immorlica, K. Jain, M. Mahdian, and K. Talwar, "Click Fraud Resistant Methods for Learning Click-Through Rates," in *WINE*, 2005, pp. 34–45.

[19] A. Metwally, D. Agrawal, and A. El Abbadi, "DETECTIVES: DETEcting Coalition hiT Inflation attacks in adVertising nEtworks Streams," in *WWW*, 2007, pp. 241–250.

[20] A. Metwally, F. Emekci, D. Agrawal, and A. El Abbadi, "SLEUTH: Single-pubLisher attack dEtection Using correlaTion Hunting," in *VLDB*, 2008, pp. 1217–1228.