

Towards Learning Intentions in Web 2.0

Gregory Blanc * and Youki Kadobayashi

Graduate School of Information Science, Nara Institute of Science and Technology
Takayama 8916-5, Ikoma, Nara, Japan
{gregory,youki-k}@is.aist-nara.ac.jp
<http://iplab.aist-nara.ac.jp>

Abstract. Web attacks are plaguing the Internet and there is no single day without a user getting infected, subverted or trapped into giving away private or bankable information. Many intrusion systems have tried to protect Web servers but eventually fail to tackle the scope of Web 2.0 attacks. Browser plug-ins are sure a good alternative for preventing the user from being the victim of untrusted Web sites but are often specialized on a single class of attacks. Eventually, rare are systems that can block 0-days. Our proposal tends to abstract the Web experience to the level of intentions. Given an intention, we expect to disambiguate malicious from benign traffic, ensuring the user's security. There is a potential to capture a large spectrum of intentions by using machine learning methods that are usually used in behavior anomaly detection. Also recent results in data mining have drawn interesting results in the structure of JavaScript codes that are the core of Ajax-based attacks. Assuming that intention is the aggregation of behaviors captured at different levels, we plan to intensify works in distributed learning systems that can correlate such intentions. Such research work should attract the attention of the different communities and seek cooperation since it will be valuable for both research domains.

Key words: intention, Web 2.0 security, machine learning

1 Introduction

The cyber security community has been tracking the trends of cyber criminality from years. Especially, since the second half of the 2000s, the spread of rich-Internet applications, social networks and Web Services have increased the attack space usable for attackers. Recently, along with the success of phishing sites, more and more attackers are now concentrating on the end user for several reasons: it requires less technical knowledge, may be easier to exploit and is far more successful. Symantec, in their 13th Internet Security Threat Report[1], have clearly identified that shift towards end users. On the other hand, WebSense has also highlighted the fact that now most of the top 100 Web sites cannot be

* The author is supported by the Lavoisier program of the French Ministry of Foreign and European Affairs.

trusted since 70 of them contained malicious contents or redirections over the last year[1].

Starting from the point that the browser must also be able to ensure the user's security, we thought about a way to detect these malicious purposes that inhabit the malicious contents that plague the Internet. In artificial intelligence, it is possible to infer the intention of an agent based on its behavior and possibly predict future actions or outcomes of present actions. Intention-based systems bear the potential to detect attacks that are difficult to correlate with the actual solutions. To enhance security solutions, several technologies can be employed such as machine learning methods and multilayer tracing techniques.

The reminder of this paper is as follows: Section 2 will introduce the concept of intention while technologies and related works concerning the Web 2.0 security and machine learning methods will be developed in Section 3 and 4 respectively. We will finally explain our proposal in Section 5 and conclude in Section 6.

2 The Concept of Intention

Simply put, intention is a concept that describes the purpose of an agent to fulfill a goal and to stick to this plan unless later reconsiderations push him to give up on actually realizing the goal. The notion originates from philosophy and has been studied and formalized within the well-known model of Belief-Desire-Intention (BDI)[7]. This model was the basis of an extensive research in artificial intelligence, especially agent theory, among which one of the most influential contributions is the idea of Cohen and Levesque that *intention is choice with commitment*[12]. In their article, they develop a formal theory for rational behavior that revolves around the BDI model. Intention of an agent x to perform an action p is then defined as the persistent goal of the agent x to reach the state of believing that it will realize the action p and eventually it does (cf. Fig. 1). A persistent goal is defined by the belief that the action p is false, i.e., not realized, to start with. As this first assumption holds, agent x chooses worlds in which action p will be later performed. Agent x is committed to performing action p unless it drops on the idea of performing p because he believes either that p is achieved or that p will never be. Later works in agent

$$(\text{INTEND } x \ p) \equiv (\text{P-GOAL } x \ [\text{DONE } x \ (\text{BEL } x \ (\text{HAPPENS } p))]; \ p)$$

where:

$$(\text{P-GOAL } x \ p) \equiv (\text{BEL } x \ \neg p) \wedge (\text{GOAL } x \ (\text{LATER } p)) \wedge [\text{BEFORE } [(\text{BEL } x \ p) \vee (\text{BEL } x \ \neg p)] \neg(\text{GOAL } x \ (\text{LATER } p))]$$

Fig. 1. Formal definition of the intention of an agent x to perform an action p .

theory have focused, among other research goals, on how machine agents could

mimic another agent behavior and especially what role intention can play in the agent behavior. However, discerning intention in human agents is not trivial and raises several issues[4]:

- what kind of information about intentions is actually available in the surface flow of agents' activity?
- which aspects of this structure can be detected?
- what kinds of additional information, if any, might be needed to account for inferring intentions and purposes of agents' activity?
- how skills can be acquired to infer intentions?

Baldwin and Baird[4] also outlined the fact that humans, mostly adults, are able to process continuous action (activity) streams in terms of hierarchical relations that link smaller-level intentions with intentions at higher levels. This is quite challenging for machines at the current state-of-the-art. Nonetheless research works in imitation have later showed that actually understanding intention was possible for an artificial system. Using a learning method in which the imitator keeps track of the intentions of the initiator, the imitator is able to reproduce a behavior after repeated trials[19]. The computational model relies on a blocks world in which goals are expressed as relations between the blocks. Although the contextual environment is absent from their experiment, they were able to achieve a good imitation in terms of goals. They did it by focusing solely on the goal, without enforcing the spatiotemporal order of the different steps. But what works well for imitation in robots may not be applicable elsewhere. Especially, when dealing with computer systems, it seems on the contrary that a particular set of steps, ordered in space and time, may define different intentions toward a same goal, since an agent may reconsider fulfilling a goal.

To account for that fact, successful incursions of intention can be found in computer security literature. For example, BINDER[13] is a host-based extrusion detection software that leverage the intention of a user to detect malware activity. The detection engine needs no prior knowledge of the system and infers user's intention from the user's activity. Mouse and keyboard events are collected and mapped with processes' activity. Assuming a process is likely to be active shortly after receiving user input, processes of which behavior deviate this simple rule may be seen as anomalous and thereby not user-intended, which is more likely to be an extrusion attempt from a malware. The system is also able to learn from false positives and to correct the threshold time for a given process, i.e., the delay time a process should not exceed after user inputs to be considered as user-intended.

To summarize, there is space to develop more techniques to infer intentions demonstrated by programs. Just as there is a potential to infer intentions of humans[4] based on external demonstrations of intention, i.e., the human agent behavior, we believe it is possible to understand, in a similar way, intentions of automated machine agents, especially computer systems, processes or event automated scripts. By doing so, one can gain insights on a process without prior knowledge or detect anomalies by learning. Another approach would be similar to [13] by coupling machine behavior and user behavior. This is appropriate for

machine-user interactions, especially Web experience in which a user interacts with a server through a browser. We believe this offers a new point of view in tackling agent behaviors through a spectrum of intentions, among which malicious ones.

3 Web 2.0: The New Playground for Attackers

There is no strict difference between Web 1.0 and Web 2.0 but it is universally understood that Web 1.0 applications rely mainly on the HTTP protocol to download pages in a synchronous pattern. On the other hand, Web 2.0 applications do involve abundant processing on the client side through embedded scripts transferring data to the server, even asynchronously, without the user experiencing delays. Web 2.0 does not rely on any particularly recent technology but on technologies that have been spreading the Web since its early years. JavaScript and XML, at the origin of the coined word Ajax (Asynchronous JavaScript with XML, 2005)[1], were technologies designed in the mid-1990's. However what characterize Web 2.0 applications are their content-richness, their collaboration features (user participation, folksonomies, social networks), their ability to syndicate contents (aggregate sites, RSS/Atom feeds, mashups) as well as their extensive use of the Ajax framework to perform dynamic, asynchronous HTTP transactions. Other essential objects comprise the Document Object Model (DOM)[1], which is usually modified dynamically to avoid reloading web pages, JS Object Notation (JSON) objects[1], which are used for the serialization of structured data during XHR[1] object transactions. Further information can be found in [18].

3.1 JavaScript Security

JavaScript (JS), which is the de facto standard in the Ajax implementation, is a dynamic, interpreted language that runs in browsers, and also locally in some applications, e.g., PDF files[1]. It is a procedural event-driven prototype-oriented language that can modify a Web page through the DOM. It can load remote files or scripts that will then be executed.

Natively, JS language does provide some security basic mechanisms such as:

- the Same-Origin policy[1] which prevents Web pages from different domains to access each other's contents and state;
- the JS language's core library is sandboxed in the browser environment since it does not implement utilities to have access to local files or to perform networking tasks;
- the signed script policy[1] in which only scripts that are digitally signed are granted permission such as execution.

However, JS is highly dynamic, allowing anyone to inject code during runtime. This can be harmful to a system by either of the following:

- exploiting an unvalidated user input vulnerability, it can trigger one of many traditional attacks: resource enumeration, parameter manipulation, especially SQL injections, XPath injections since the introduction of the XPath language in the DOM level 3, XSS-based attacks[1] (in particular the DOM-based flavor[1]), session hijacking, etc.
- clobbering JS functions, the purpose or the control flow of a script can be modified maliciously;

Further information on JS can be found in [15]. The reader may refer to [1] for a tutorial on JS security issues.

3.2 Not a Vulnerability but a Feature

As we saw, vulnerabilities are not linked to the JS language per se, but rather to its quick spread among developers that rarely had the time to master all the subtleties of the language, leading to many vulnerabilities, e.g., the way flow is handled on the client-side or the way data are validated on the client-side.

The XMLHttpRequest (XHR) object[1] is the core element of the Web 2.0. It allows for seamless Web application experience by handling asynchronous connections. This mechanism when subverted does not only allow generating request-embedded attacks, but also reading HTTP responses and bodies within the Same-Origin policy[1]. This leads to the development of automated JS programs, that can be used in botnets.

Ajax applications do present some security challenges[1]:

- the transparency has been increased by the exposure of the application logic and security, as well as the server methods;
- the circumvention of the browser security measures: for instance, XHR responses are directly processed by JS, and if these contain any script code, it would be processed all the same without any character encoding or other security measures that could be enforced by a browser; There are also ways to circumvent the Same-Origin policy through the use of legitimate `` or `<style>` tags making cross-domain requests, or by exploiting conception or implementation vulnerabilities;
- the increase of the attack surface: the traditional Web application attack surface (form inputs, cookies, headers, hidden form inputs, query parameters and uploaded files), the client-side JS code, the APIs' flaws and the exposed Web Services;
- the ability to manipulate other Web components such as Active X controls, Flash contents or Java applets, and have access to local files or launch network attacks;
- large modular applications and the relative invisibility of XHR transactions have increased the complexity. Ajax vulnerabilities also lie in the manipulation of untrusted sources' scripts. Especially, mashups[1] are such websites that act as Ajax proxies and allow bypassing the Same-Origin policy. In [5], Barth et al. demonstrated that the mashup communication incurs real risks of being compromised.

3.3 Countermeasures

The main vector of the Web application vulnerabilities is obviously the lack of input validation, leading to many common injection attacks. To sanitize or validate these inputs, it is necessary to first locate these. Usually, user inputs comprise visible form inputs as well as hidden form inputs, HTTP headers, including cookies, URLs especially the query string and its parameters, as well as POST parameters that are present in the HTTP request body, and uploaded files. As pointed out, the attack surface also includes the Web Services, therefore Web Service parameters should also be properly validated.

(Black|White)list The first and most straightforward way to block scripts injected in the vulnerable inputs is to actually try to recognize signatures of malicious scripts, i.e., to match a short string specific of the malicious script. However, the diversity of attack vectors and the fact that they evolve, as time passes, forces the developers to constantly update their blacklist (BL) which is inflexible. If they fail to, evasion is more likely. This method is rather reactive and does not protect against unknown attack vectors. On the contrary, the whitelist (WL) focuses on the vulnerability itself rather than on the threats. It features a list of accepted values or expressions to be found on legitimate inputs. Their absence entails rejection of the input. However, building a WL is more complex and combining the two approaches has generally proven to be more efficient.

Rich user input validation Things get even more complicated when validating rich user input. Indeed, these inputs may actually accept scripts or at least XML-like tags as legitimate values, which incurs that attacks may go undetected. Proper validations do include among others[17]:

- markup language validation: compliance with a scheme or a protocol as for the structure and the type of inputs, contents may also be subject to WL;
- binary file validation: as any other type of inputs, binary files structure, size and type of the data contained in files are as much input as to be validated possibly using WLs but one can also discard unrecognized structures;
- JS validation: when scripts are an accepted value type, it is not trivial to tell whether a JS code is malicious or not[1]. The difference lies much more in the context these functions are actually used. Indeed, a same function can perform in a legitimate way and in a way that in the end is malicious;
- JSON: subject to JSON Hijacking[1] which can be avoided using an infinite FOR loop to evade a hijacking script[17].

If we refer to current books on the subject[17, 31, 9], there is really not much more to do to protect Web 2.0 applications, along with minimizing data leakage on the client-side.

Malicious JS The challenge is to determine which scripts are not malicious and which scripts are. This is not trivial, JS being a highly dynamic language: e.g., a script that, when loaded, may look innocuous at first, can later modify itself or be modified to perform unwanted actions. JS function clobbering[1] is a good example of such attack. JS scripts are also polymorphic, i.e., there are many ways to achieve one action, and different notations can be employed as well as obfuscation[1] and encoding techniques, frequent among attackers. This makes the static analysis a really difficult task and matching malicious patterns through regular expressions quickly becomes useless. Current approaches are mostly manual and intensive, thereby not applicable to great scale applications. Nonetheless, recent tools such as the offline static behavior analysis tool Caffeine Monkey[1], or the online Jsunpack[1] static deobfuscator produce very interesting and accurate results. Still it fails to cover the entire spectrum of vulnerabilities provided by malicious scripts. A more straightforward attempt was made by Chenette[1] who hooked Microsoft's Internet Explorer DLL files to obtain a non-rendering browser that would output interesting browser security events.

About Security Devices When it is not possible to review the source code and to correct the vulnerabilities, security may rely on external security devices such as firewalls, IDS/IPS and lately Web Application Firewalls (WAF). While the formers are inadequate to process data at the application level since these devices are traditional network security devices, the latter have proven to be successful at blocking traditional Web attacks since it has been built from the grounds of the HTTP protocol. However, WAF may be used as a temporary solution to give time to developers to fix the vulnerabilities discovered in a Web application since present protections may be bypassed in the future. This process is called Just-in-Time Patching (or Virtual Patching)[1]. And if the prior assumption that we are not able to modify the application code holds, our Virtual Patch will need to be kept updated. WAF also suffer some downsides such as the fact that it does not provide protections against client-side attacks such as DOM-based XSS which is performed completely on the client-side with the attack payload never reaching the server-side.

To summarize, Web 2.0 applications' complexity makes design and deployment of countermeasures a difficult task and transparency literally disclose the Web application client-side JS logic and flaws. In the middle, the end-user has no one to trust and is at risk when simply surfing the Web 2.0.

Browser extensions and plugins Aside from widely adopted browser extensions/plugins, especially for Mozilla browsers, that allow to block or disable scripts[1], academic research have produced a number of systems, whether embedded in the browser or deployed as an external module or proxy, to prevent compromises of the browser and especially its state, the user's session information or credentials. The downside is that most of these tools perform only specialized tasks to minimize the processing and networking overheads that the

browser incurs. This leads to a myriad of tools ranging from anti-CSRF tool[6], to input validation[1], sanitizing proxies[1, 27], policy enforcement[20].

There is therefore a need for a solution that allow for detection of a wider scope of attacks. What if it was possible to tell that a certain request, a certain script code bears a malicious intention? What if when analyzing a certain request or script, we were able to predict whether it will do good or bad to a user browsing a vulnerable Web application?

4 Machine Learning in Cyber Security

The complexity of the current computer systems and the forthcoming ones makes it difficult to design security policies that are simple to understand and flexible to tolerate. Also, attacks are constantly evolving and mutating, which should be taken into account to defend against such threats. Machine learning methods have already been successfully applied to computer security issues for some years, often in a bid to detect technical attacks originating from the outside network. Less specifically, while computer security revolves around 3 main activities, namely prevention, detection and recovery, machine learning and data mining techniques are mostly applied to detection, but may also be used successfully for other activities.

Actually much of the machine learning works in the cyber security domain has targeted the layer of network and host-based intrusion detection systems (IDS). IDSs are usually of two kinds: based on detecting signatures or based on anomaly detection. Machine learning is good at building models from which it is possible to infer a behavior deviation from normal activity, called anomaly. Machine learning techniques enable the development of anomaly detection algorithms that are non-parametric, adaptive to changes in the characteristics of normal behavior in the relevant network, and portable across applications[3]. However these systems often suffer from high false-alarm rates and a relatively poor coverage of the attack space. It is often due to the sources of information that are used to infer anomalies and which may be biased or incomplete.

4.1 About Machine Learning

Machine learning is the science of building predictors from data randomly sampled from an assumed probability distribution while accounting for the computational complexity of the learning algorithm and the predictor's performance on future data[29]. Machine learning methods allow gaining some insights on a set of data labeled in different classes (examples) or unlabeled (observations).

Definitions Learning algorithms have 3 components[26]:

- the representation (or hypothesis language, or concept description language) is a formalism used to build models. A variety of representations have been used: trees, rules, graphs, probabilities, first-order logic, coefficients of linear

and nonlinear equations, or the examples themselves. A model (or hypothesis, or concept description) is a particular instantiation of a representation. In most cases, models generalize examples and can then be used to make predictions about examples absent from the original set;

- the learning element builds a model from a set of examples;
- the performance element applies the model to new observations.

Once a set of examples has been formed, learning algorithms can support various analysis tasks:

- anomaly detection: a model is built from all examples and anomalous events or observations are detected;
- classification: a model built using 2 or more classes in which examples are distributed. New observations are then classified as being of one class or another. Two-class problems are referred to as detection task, with class labels of positive and negative;
- regression: in this case, prediction is not made on one of a set of values but rather for a numeric value;
- association rules: examine the associations between sets of attributes.

Computational Learning Theory (COLT) It is the mathematical field related to the analysis of machine learning algorithms. Learning is one of the most difficult intellectual skill to computerize. At the current state-of-the-art, 4 different types of activities fall under the machine learning rubric[29]: symbol-based, connectionist-based, behavior-based, immune-system-based. Please refer to [1] for extensive literatures on COLT.

4.2 Previous Incursions

There is already an impressive and extensive literature on the applications of machine learning methods into the computer security world. More generally, many attempts have been made to design anomaly detection systems using different learning methods. Examples comprise a time-based inductive learning machine for anomaly detection[32], a road traffic incident detection system[1] and lately a demonstration of an all-purpose network anomaly detection using different machine learning methods[3]. These works have demonstrated effective result and motivate the development of more advanced, fully adaptive (that can adjust to changing environments in the process of learning), fully distributed (that can relax the learning overhead over several components) learning algorithms.

Since machine learning methods are more fitted for detection activities, it was obvious that many attempts would tackle intrusion detection at different levels on the computer or on the network[33, 1, 28] and more recently to detect worms[2] and Web attacks[16]. This last article, though making an incursion into Web application attack detection, does not consider in any way Web 2.0 applications. Attacks mitigated in this research work are well-known and fully documented.

4.3 Learning Methods to Detect Web Attacks

Quite few contributions in Web application security dared to employ machine learning methods to perform detection of known and unknown Web attacks. The first one was already quite elaborated in the sense that it proposed several models directed at different attributes or set of attributes derived from the observation of normal requests[21, 22]. They assume that attack patterns differ from normal behavior and that these differences can be expressed quantitatively. They assume normal requests always present the same parameters in the same order and leveraging that fact, their learning method builds features on top of parameter attributes in order to obtain a normal behavior. This work prefigured what is implemented in WAFs today and already advocate the idea of building profiles for custom-developed web applications, which lists the different URL paths. Interestingly they encountered detection of false positives during the training period although most of these were due to search engine user input mistakes.

Concerning the attack detection part, their tool reliably detected 11 real-world exploits downloaded from security-related websites, all of these were traditional Web attacks. No 0-day exploit was tested though. The conference article[21] was later updated as a journal article[22] which presented 3 new models considering the access frequency of a website, the inter-request delay time and the invocation order of the different programs that compose a Web application. But they did not test the capacity of these 3 models to detect Web-based attacks. Robertson et al.[30] later pursued the same research works and addressed several drawbacks inherent to anomaly-detection systems which are their proneness to a high number of false positives as well as the lack of information about the type of attack that has been detected. To address the former, they developed an anomaly generalization technique that automatically generates signatures from anomalous events to allow a more abstract representation of an event and to group similar events. To address the latter, they developed an attack class inference component that matches anomalous events with sets of heuristics that allow classifying these into 4 distinct classes: directory traversal, XSS, SQL injection and buffer overflow. They evaluated the attack detection component against the same set of attacks as used before, to which they added SQL injection attacks.

Starting from the same observations as Krueger et al.[21, 22, 30], Estevez et al.[14] have modeled the behavior of the HTTP protocol and applied it to anomaly detection. Using a Markov model, they are able to enforce proper use of an application along with HTTP specifications. Each HTTP request is then parsed against their Markovian model which has been priorly trained using normal traffic to the specific Web site. They subsequently evaluated their scheme using the DARPA 1999 IDS Evaluation Program datasets[1]. However, in this article, they were not able to tackle issues as the need to re-train their model as the Web application evolves. Similarly, Cheng et al.[11] proposed their Embedded Markov Model (EMM) to not only detect anomalies in the user inputs that violate the Web application model but it can also detect an unreasonable page transition in the Web application flow.

The EMM performs in two distinct phases: first, the lists of all distinct pages from a Web site and attributes are extracted. These strings are then parsed to draw the probabilities of transition between one string to another: i.e., all the transition paths and all the reasonable values for all attributes. Second, the User Behavior Template (UBT) is built upon the probabilities of page transitions and the probabilities of having a certain set of attributes for a given page. Here also it is questionable whether such a scheme fits the requirements of a dynamic URL website or simple websites that do evolve quickly.

5 Learning Intentions in Web 2.0 Applications

Against the current of what has been initiated in the machine learning field, we are not focused on protecting the Web server. Symantec Internet security report[1] has witnessed a shift towards client-side (browsers) issues from both the security research and attack communities. WebSense Security Labs' State of the Internet 2008[1] includes 8 client-side vulnerabilities in their Top 10 Web attack vectors ranking, among which browser vulnerabilities are ranked first, SQL injections third and malicious Web 2.0 components fourth. This report also features the following highlights:

- 77% of Web sites with malicious code are compromised ones;
- the number of malicious Web sites identified by Websense Security Labs from January first, 2008 through January first, 2009 has increased by 46%;
- 70 of the top 100 sites either hosted malicious content or contained a masked redirect to lure unsuspecting victims from legitimate sites to malicious sites. This represents a 16% increase over the last six-month period.

5.1 A New Target: The User

So the fact that the Web 2.0 end-user is the most at risk is no more a figment of one's imagination. The multiplication of phishing-related attacks[1], client-side-only XSS attacks[1] such as DOM-based XSS or XSS tunnels[1], attacks on Social Networks[1] such as OSRF/CSRF, CSS/HTML/DOM modifications, botnet's zombie harvesting, local network attacks using the user's browser as a stepstone. These are due to Social Networks for loosely allowing a lot of users' contributions and links to outside untrusted websites, and for not implementing enough security in the process.

We argue that necessarily we need to protect the user since even protecting websites will not prevent a user being trapped in visiting malicious Websites. There is thereby a need to find ways of tracing code running in the browser, detecting unwanted connections from and to the browser as well as communications between the browser and the user's system, detecting and blocking malicious scripts and notifying the user of possible threats. It is understood that more experienced users are not at risk but the spread of Internet connectivity, especially in developing countries, increases the number of novice users. Not

anyone does have interest in computer science or information security and some people begin experiencing the usage of computers only when at school or work. The key here is that these users did not intend actions that were taken against legitimate websites or against themselves. If we are able to infer intentions of users, we can prevent them from being tricked. On the other hand, malicious attacks and scripts do bear intentions that are different from what the user would usually intend. If we can infer that the actions taken on the browser bear intentions different from the user's intention, we can therefore take actions to prevent further compromise.

5.2 Leveraging Intentions

How can we infer intentions? Human intentions can be conveyed by external signs i.e., the behavior. We believe machine intentions can also be inferred from behaviors. Anomaly detection is such way to detect differences in normal and abnormal behaviors. We argue that there is an added value to associate the user and the machine behavior to detect malicious intentions. Such correlation has the potential to reduce false positives that is one of the downside of anomaly detection.

Especially, in cases which the user can not easily repudiate actions performed on his behalf, such as in CSRF, where a user being tricked in surfing a malicious Web page would unintentionally be issuing requests against another Web application of which session is still active. Here the user's goal is to surf the malicious Web page or another Web application, which he does. But without any other action from the user, unintended requests are executed on another Web application, which, in terms of intentions, sounds malicious.

On the other hand, there are also cases in which intention may fail to accurately identify an attack. For example, a user tricked into visiting a malicious Web page in which she intently download a file, i.e., clicking on a link, can not repudiate her intention since the request was explicit and not silent as in an XHR request which does not require the user to perform any action. Generally, attacks that are performed in a single step with a goal that is not different from the user's intention can be tricky to detect and may bypass an intention detection system. Here we can feel, that necessarily, intentions are built on a structured behavior that may unfold in steps and comprise an identified goal. Failing on identifying this goal may lead to the ineffectiveness of this proposal in certain conditions.

To build such behavior, we believe that actual researches have paved the path but obviously ignored Web 2.0 related issues and are not completely applicable in our research. Previous exhibited behaviors were focused on building static WLS against conventional attacks and their vision can not be understood here where the intention is not binary: there is not one good intention and one evil intention. There are plenty of intentions, among which *grey* ones. Previously used features are not sufficient to represent the large scope of intentions and therefore we need to dig deeper to find and experiment new set of features. Recently, research works in the analysis of JS codes have offered some tools like Monkey Caffeine[1]

to statically analyze malicious JS codes. Using this tool, they demonstrated, for a set of JS codes they gathered, that the ratio of function calls is similar between benign scripts and similar between *bad* scripts, and also that this ratio is different between the two aforementioned subsets. We may reproduce this work to seek confirmation of such phenomenon and therefore go on to experiment this research on a larger scale and finally build the intention-system on such grounds.

Alternatively, research in JS categorization has also progressed leading to semantic tools that allow subsequent classification of code samples[23–25]. But we can also think of many other ways to gather new metrics. In particular, we propose tracing the user behavior and the browser processes and scripts. For that, we may use such tools as Sun’s dtrace[10, 1] or VMWare’s DynamoRIO[8, 1] which are relatively recent tracing frameworks that allow the development of very precise tracing scripts that will allow drawing new metrics, inexperienced in Web security. Especially dtrace allows to run different tracing scripts that can cooperate via the use of context variables and thereby makes correlation of different events possible.

5.3 Leveraging State-of-the-art Technologies

It is universally understood that, on one hand, distributed systems perform better than monolithic systems, and, on the other hand, ensemble learning provide more accuracy than a single classifier. We argue that a system leveraging both previous statements can outperform existing learning detection systems and is much more fitted for the client-side since: (1) distributing allow reducing the processing overhead. We can distribute part of the processing to a proxy higher on the local network or to an external module, communication to the browser therefore needs to be secured; (2) ensemble learning will leverage the correlation of different metrics harvested upon the instrumented processes and sensors, and allow to have more adapted learning methods for different features.

6 Conclusion

In this paper, we have highlighted several points concerning how security in Web 2.0 applications was handled today and how we could probably tackle this problem in a more thorough and uniform way, especially:

- computer security have provided solutions that work for traditional Web applications but still fail to keep up the pace with the growing number of attacks and variations made possible by the usage of dynamic asynchronous scripts;
- Web 2.0 through the Ajax framework have modified the way attacks are carried out, and the attack model has shifted from targeting the Web servers to targeting the user, which is a more vulnerable target that can be easily lured. The motives of the attackers have thereby shifted from subverting a Web server to stealing a user’s session, stealing a user’s private information

- or credentials, subverting the user's machine in the process of building a botnet, using the user's machine as a stepstone in the process of intruding a private local network;
- machine learning has already been used successfully in tackling computer security issues and many schemes revolving around intrusion detection or anomaly detection have been developed;
- detecting a human being's intention is possible by observing its behavior but a context is also necessary for finely determining among a set of diverse intentions (more than two). Observing the Web 2.0 traffic in a browser may denote possible intentions born by requests and responses, or by a set of multiple transactions;
- there is a potential to apply machine learning to learn intentions of legitimate traffic, to learn intentions of specific Web attacks, to learn intentions of legitimate scripts, to learn intentions of attacks scripts, etc. This induces a computational overhead that may be tackled using a distributed system at different layers and by instrumenting the browser using optimized tools.

In a bid to achieve this goal, we believe that researchers in cyber-security and machine learning, as well as artificial intelligence researchers, have to collaborate. Each of these disciplines can feed each other with advances in their own domain.

References

1. http://iplab.aist-nara.ac.jp/~gregory/jwis_annex.html. Due to the limitation of pages, you can find these references on our online annex.
2. J. Agosta, C. Diuk-Wasser, J. Chandrashekar, and C. Livadas. An Adaptive Anomaly Detector for Worm Detection. In *2nd Workshop on Tackling Computer Systems Problems with Machine Learning Techniques*, 2007.
3. T. Ahmed, B. Oreshkin, and M. Coates. Machine Learning Approaches to Network Anomaly Detection. In *2nd USENIX Workshop on Tackling Computer Systems Problems with Machine Learning Techniques*, 2007.
4. D. Baldwin and J. Baird. Discerning intentions in dynamic human action. *TRENDS in Cognitive Sciences*, 5(4):171–178, 2001.
5. A. Barth, C. Jackson, and W. Li. Attacks on JavaScript Mashup Communication. In *Web 2.0 Security and Privacy*, 2009.
6. A. Barth, C. Jackson, and J. Mitchell. Robust Defenses for Cross-Site Request Forgeries. In *ACM Conference on Computer and Communications Security*, 2008.
7. M. Bratman. *Intention, Plans, and Practical Reason*. CLSI Publications, 1999.
8. D. Bruening, T. Garnett, and S. Amarasinghe. An Infrastructure for Adaptive Dynamic Optimization. In *IEEE International Symposium on Code Generation and Optimization*, 2003.
9. R. Cannings, H. Dwivedi, and Z. Lackey. *Hacking Exposed Web 2.0: Web 2.0 Security Secrets and Solutions*. McGraw-Hill, 2007.
10. B. Cantrill, M. Shapiro, and A. Leventhal. Dynamic Instrumentation of Production Systems. In *USENIX Annual Technical Conference*, 2004.
11. Y. Cheng, C. Laih, G. Lai, C. Chen, and T. Chen. Defending On-Line Web Application Security with User-Behavior Surveillance. In *3rd IEEE International Conference on Availability, Reliability and Security*, 2008.

12. P. Cohen and H. Levesque. Intention is Choice with Commitment. *Artificial Intelligence*, 42:213–261, 1990.
13. W. Cui, R. Katz, and W. Tan. BINDER: An Extrusion-based Break-In Detector for Personal Computers. In *USENIX Annual Technical Conference*, pages 363–366, 2005.
14. J. Estevez Tapiador, P. Garcia Teodoro, and J. Diaz Verdejo. Detection of Web-based Attacks through Markovian Protocol Parsing. In *10th IEEE Symposium on Computers and Communications*, 2005.
15. D. Flanagan. *JavaScript: The Definitive Guide, 5th Edition*. O’Reilly Media, 2006.
16. V. Garcia, R. Monroy, and M. Quintana. Web Attack Detection Using ID3. In *IFIP 19th World Computer Congress on Professional Practice in Artificial Intelligence*, pages 323–332. Springer, 2006.
17. B. Hoffman and B. Sullivan. *Ajax Security*. Addison-Wesley (Pearson), 2007.
18. A. Holdener. *Ajax: The Definitive Guide*. O’Reilly, 2008.
19. B. Jansen and T. Belpaeme. A Computational Model of Intention Reading in Imitation. *Robotics and Autonomous Systems*, 54:394–402, 2006.
20. T. Jim, N. Swamy, and M. Hicks. Defeating Script Injection Attacks by Browser-Enforced Embedded Policies. In *ACM International World Wide Web Conference*, 2007.
21. C. Kruegel and G. Vigna. Anomaly Detection of Web-based Attacks. In *10th ACM Conference on Computer and Communications Security*, 2003.
22. C. Kruegel, G. Vigna, and W. Robertson. A Multi-Model Approach to the Detection of Web-based Attacks. *Computer Networks: The International Journal of Computer and Telecommunications Networking*, 48:717–738, 2005.
23. W. Lu and M. Kan. Supervised Categorization of JavaScript using Program Analysis Features. In *Asian Information Retrieval Symposium*. Springer-Verlag, 2005.
24. S. Maffei, J. Mitchell, and A. Taly. An Operational Semantics for JavaScript. In *6th Asian Symposium on Programming Languages and Systems*. Springer-Verlag, 2008.
25. S. Maffei and A. Taly. Language-Based Isolation of Untrusted JavaScript. In *22nd IEEE Computer Security Foundations Symposium*, 2009.
26. M. Maloof, editor. *Machine Learning and Data Mining for Computer Security*. Springer, 2006.
27. A. Moshchuk, T. Bragin, D. Deville, S. Gribble, and H. Levy. SpyProxy: Execution-based Detection of Malicious Web Content. In *16th Annual USENIX Security Symposium*, 2007.
28. A. N.B., S. Benferhat, and Z. Elouedi. Naive Bayes vs Decision Tree in Intrusion Detection Systems. In *ACM Symposium on Applied Computing*, pages 420–424, 2004.
29. V. Rao Vemuri, editor. *Enhancing Computer Security with Smart Technology*. Auerbach, 2006.
30. W. Robertson, G. Vigna, C. Kruegel, and R. Kemmerer. Using Generalization and Characterization Techniques in the Anomaly-based Detection of Web Attacks. In *13th Symposium on Network and Distributed System Security*, 2006.
31. P. Stuttard and M. Pinto. *The Web Application Hacker’s Handbook: Discovering and Exploiting Security Flaws*. Wiley, 2007.
32. H. Teng, K. Chen, and S. Lu. Adaptive Real-time Anomaly Detection Using Inductively Generated Sequential Patterns. In *IEEE Computer Society Symposium on Research in Security and Privacy*, pages 278–284, 1990.
33. A. Valdes and K. Skinner. Adaptive, Model-based Monitoring for Cyber Attack Detection. In *Recent Advances in Intrusion Detection*. Springer, 2000.