

Sponsored by:

NETWORKWORLD

This story appeared on Network World at
<http://www.networkworld.com/news/tech/2011/060611-cyberattack-phases.html>

Insider's view into Web application abuse: 5 phases of an attack

By Kyle Adams, chief architect, Mykonos Software, Network World
June 06, 2011 05:10 PM ET

This vendor-written tech primer has been edited by Network World to eliminate product promotion, but readers should note it will likely favor the submitter's approach.

Sponsored by:

With [congressional hearings](#) on data theft following the [Sony PlayStation data breach](#) of 100 million records, and news from companies such as [Epsilon](#), [HBGary](#), [RSA](#) and [Barracuda Networks](#) about recent cyberattacks, [security](#) of data in an online, interconnected world has become a mainstream topic.

Gartner claims that 70% of all vulnerabilities are at the Web application layer. Indeed, the majority of attacks today, including the Sony PlayStation attack, involve some form of Web application vulnerability. But an application-related data breach is not just a one-time, isolated event. The actual breach is only one stage of the attack.

ANALYSIS: [Key lessons learned from Sony hack-fest](#)

Here are the five distinct phases of an attack on a Web application.

Phase 1: Silent reconnaissance

The attacker gathers as much information as possible identifying potentially vulnerable areas of the application. This is done discretely using tools such as Web debugging proxies to monitor the traffic between the browser and the Web [server](#). The attacker traverses the site, much like a normal user, while collecting valuable information about how the application works. This activity goes undetected, because as far as the server is concerned, it represents the traffic of a legitimate user.

At this point, the attacker will stop interacting with the target server directly. The attacker will spend significant time reviewing the data collected by the debugging proxy and extracting useful facts about the environment. This

may include the type of hardware and software in the network architecture, programming languages, libraries, source code and comments. This information will be leveraged during the later phases of the attack.

Phase 2: Attack vector establishment

This phase begins once the attacker has gained an understanding of the application design and the breadth of its attack surface. Until now, the interaction with the server has been fairly benign and undetectable, but in the next phase, things get a little louder. For this reason, the attacker will often start using an [anonymous proxy](#) to interact with the server.

The attacker may also employ other protective measures such as browser privacy controls, firewalls, antivirus and virtual machines. Once the attacker is confident that his traffic can no longer be traced, the real work can start.

With notes in hand, and a debugging proxy up and running, the attacker starts to seek out dynamic pages, especially those which accept form or query input. The attacker will then determine what the various input parameters are, and attempt to derive boundary cases for them. Boundary case values are sent to the application to provoke an unintended response from the server.

The attacker repeats this activity on all dynamic pages that he is aware of. When finished, he has a list of all the parameters that are correctly validated by the server, and more important, the parameters that are vulnerable -- they produce calculation errors, fatal errors, or are blindly injected into the response without encoding or cleansing.

The attacker tailors the boundary cases so they do not match any known attack signatures, so this activity is almost always imperceptible to server administrators. The attacker still has to remain anonymous, because many [applications](#) keep track of errors and record the addresses of the clients responsible for generating them. Because of this, administrators could discover the activity later by inspecting logs with a security tool. However, this is typically long after the attacker has moved on to the next phase.

If the attacker was able to obtain a large number of potentially vulnerable inputs, the next step is to start testing each one to see if an attack vector is possible. For example, if the attacker received an SQL error when submitting a value of "my'username" in a login form, then there is probably an [SQL injection vulnerability](#). The attacker will start supplying more structured SQL syntax into the input in an effort to shape the resulting error.

At this point, attack signature detection software would likely detect the threat. However, the attacker really doesn't care if he is detected, because from the perspective of the server he is connecting from somewhere else. His goal is not to prevent detection, or even to prevent being blocked, but rather to find out what does not get detected or blocked.

If his IP address is ever completely blocked, he can just go through a new proxy. Because of the variability of syntax in any given environment, it is nearly impossible to anticipate all possible attack vectors and their permutations. The attack signature library can never be comprehensive enough. When effective attack vectors are blocked by signatures, the attacker just needs to tweak his input to avoid matching. The signature matching tool only provides another level of generic input validation that can easily be evaded.

Phase 3: Implementation

This phase begins once the attacker has identified the vulnerabilities and their associated attack vectors. This is where the real damage starts. The scope of damage depends on the types of vulnerabilities that are exploited. For example:

- The attacker starts to mine the database for sensitive information, delete existing information, or insert new

fraudulent information.

- The attacker seeds the application with malicious code by way of [XSS](#) vulnerabilities and reflected parameters.
- The attacker designs complex [phishing](#) scams that use the vulnerabilities to give the scam credibility.

The possibilities are only constrained by the potential vectors, and how they can be chained together to deliver more powerful payloads. Most of the damage has been done at this point.

Phase 4: Automation

Attacks such as input parameter abuse are often single request vectors. This means the damage happens within a single HTTP request. Sometimes, however, the execution of an attack vector provides incremental benefits each time it is performed. Generally, if the attack vector generates revenue for the attacker, the next step is to automate the attack. This enables the attacker to repeat the attack vector over and over again, multiplying the overall monetary gain.

Because the attacker must still cover his tracks in order to execute the automated attack, he will generally code the attack into a remotely controlled bot. This tactic poses serious challenges for the administrator, because even if the attack is identified, an IP-based block will no longer be sufficient. To accomplish this, attackers will often use a prefabricated "command and control" kit that allows them to quickly raise and command a bot army.

Phase 5: Maintenance

Finally the attack is complete. The hacker has extracted as much data as his experience and skill allows. He will go off and work on other projects until his automated bots start to fail. This will signal that some fundamental vulnerability in the attack vector has been patched or modified. If the attacker cares enough, he may repeat the entire process over again, focusing on the parts of the application that are essential for the bots proper functioning. He will find a work around for the new patch, create an entirely new attack vector, or move to a different target altogether.

How to protect a Web application from abuse

IT administrators are challenged to respond to Web application abuse, primarily because they can't detect when it happens. The silent reconnaissance behavior is largely invisible to traditional intrusion detection systems, and gets lost in the background noise of normal user behavior, however next-generation Web application firewalls featuring [Web intrusion prevention](#) are effective.

Administrators can deploy traditional signature-based Web application firewalls to block the actual attacks in the implementation phase. But sophisticated attackers evade content firewalls easily by tailoring their approach to avoid known attack patterns. This is made easier by the fact that the firewall filters are rarely applied restrictively, due to concerns about false positives.

Another available approach is to implement "whitelist" rules that tightly constrain all application inputs and weed out attacks. However, Web applications are complicated. It's hard to spend the time and resources getting those rules right.

The best approach over the long run is to fix the application code itself. But patching application code is also time intensive, and takes the development team away from the current deliverables. Moreover, the application code is often unavailable -- the project was outsourced, or the application is off-the-shelf from a vendor.

Today, the reality is that Web applications are mostly undefended and administrators typically find out about the attack from the symptoms and aftermath -- traffic anomalies, bursts at abnormal hours from unusual geographic

sources, and unexplained business losses.

But by this time, the attack is deeply entrenched. Administrators, like those at Sony PlayStation, are [forced to spend weeks stamping out the fire](#) -- blocking IP addresses and killing accounts -- only to have it continue to flare up.

The IT team is forced to burn their time and resources blindly, trying to combat the attack as best they can. But the business is already damaged. Worse, there are often additional infrastructure and transaction costs incurred by the automated attack traffic. Companies find themselves in the position of not only being attacked, but being forced to pay for it, too.

Isn't it time for the Web application layer to receive more security attention to prevent data theft and abuse?

Mykonos is the smartest way to secure websites and Web applications against hackers, fraud and theft. Its next generation Web application firewall detects, tags, tracks and stops hackers in real-time. Mykonos neutralizes threats as they occur, preventing the loss of data and saving companies millions of dollars from fraud or lost revenue. www.mykonossoftware.com

[Read more about security](#) in Network World's Security section.

All contents copyright 1995-2011 Network World, Inc. <http://www.networkworld.com>