2nd USENIX Workshop on Offensive Technologies (WOOT '08)

July 28, 2008
San Jose, California, USA

PAPERS

Summarized by Joshua Mason (josh@jhu.edu)

- ***Engineering Heap Overflow Exploits with JavaScript***
  Mark Daniel, Jake Honoroff, and Charlie Miller, Independent Security Evaluators

Jake Honoroff discussed a new mechanism for controlling the heap in browser-based attacks using the JavaScript engine. The technique allows attackers reliable control of the temporal deallocation of memory by forcing garbage collection. Honoroff's analysis was conducted on the WebKit JavaScript implementation, wherein garbage collection is triggered either by a timer or by necessity. The garbage collection timer in WebKit will not preempt a running script to deallocate memory. Thus, Honoroff forces the invocation of the need-based garbage collection routine by allocating large portions of memory via object instantiation and subsequently removing references to the created objects.

More specifically, attackers can force very specific heap layouts by allocating large arrays of objects and simply removing the references for any objects that need be deallocated. Then, by allocating and immediately unreferencing enough objects to trigger garbage collection, the attacker forces the deallocation of memory and has the exact heap layout necessary to complete the attack. More succinctly, Honoroff's methodology allows certain vulnerability types that previously could only be exploited unreliably to be exploited with virtual certainty.

- ***Experiences with Model Inference Assisted Fuzzing***
  Joachim Viide, Aki Helin, Marko Laakso, Pekka Pietikäinen, Mika Seppänen, Kimmo Halunen, Rauli Puuperä, and Juha Röning, University of Oulu, Finland

Joachim Viide presented work that attempts to model and subsequently fuzz file formats automatically. Naive file-format fuzzing simply generates a large number of files by flipping random bits in an input file. This approach allows the fuzzer to change fields present in the existing objects to unexpected values but not to create an invalid number of valid objects or order certain objects in an unexpected fashion. Thus, naive file fuzzing typically yields very limited code coverage.

Viide's model inference relies on automatically learning a context-free grammar from a selection of files of the speci-

fied file format. To train their models, the authors generate between 10 and 100 files by hand. The audience argued that creating these training files by hand presupposes some knowledge of the underlying file format, but the authors decided not to use a randomly harvested corpus, owing to copyright and privacy concerns.

The derived context-free grammar then allows the fuzzer to automatically generate or omit entire objects when creating files of a given format. File generation is accomplished by choosing a random probability, or "fuzz factor." The fuzz factor acts to decide, during file generation, whether a grammar rule is to be skipped, processed normally, or repeated twice. The results of their fuzzing technique were fairly impressive. The authors chose to fuzz compression/archiving file formats (e.g., ace, arj, bz2, gz, zip). They generated at most 320,000 files per file format and used them as inputs to antivirus software. These files yielded 51 unique crashes in five different pieces of antivirus software using 10 different file formats.

- ***Insecure Context Switching: Inoculating Regular Expressions for Survivability***
  Will Drewry and Tavis Ormandy, Google, Inc.

Tavis Ormandy presented work that explores the insecurities present in popular regular expression engines. Many of these engines now exist in both popular software and popular programming languages. Ormandy and Drewry designed an engine to fuzz these common regular expression engines and were able to discover vulnerabilities in SQL, PHP, TCL, Adobe Acrobat Reader, Adobe Flash, Safari, and even GnuPG.

The engine itself is written in C and attempts to generate regular expressions that will break regular expression interpreters. The process begins by randomly choosing both the expression length and the beginning term. The regular expression then expands from the inside out by randomly choosing subsequent terms. It uses feedback from GCOV, a program coverage tool used in conjunction with GCC, as input to a feedback loop that chooses paths that will trigger new portions of code. Using their fuzzer, they discovered exponential-time execution and/or compilation vulnerabilities in all tested regular expression implementations.

Because of this last revelation, an audience member asked Tavis to recommend a regular expression engine. Tavis seemed to indicate a lack of confidence in any currently available libraries but seemed hopeful about those that are being developed. He also expressed his hope that current regular expression engines would improve in the near future.

PAPERS

Summarized by Sam Small (sam@cs.jhu.edu)

- ***There Is No Free Phish: An Analysis of "Free" and Live Phishing Kits***
  Marco Cova, Christopher Kruegel, and Giovanni Vigna, University of California, Santa Barbara

Marco Cova presented a study of Internet phishing kits. Phishing is a form of identity theft in which attackers try to elicit confidential information (e.g., online bank-account information) from Internet users. These attackers, or phishers, frequently deploy Web sites that look nearly identical to legitimate Web sites, often fooling unsuspecting visitors. The information collected from such phishing attacks is frequently used to support illicit and fraudulent activity. The phishing kits examined by this study were all obtained freely from underground distribution and live phishing sites.

In particular, the study focuses on the organization and technical sophistication of phishing kits. Its results provide some insight into the current motivations and modus operandi of phishing kit authors and distributors. After surveying more than 500 phishing kits, the researchers were able to document a number of characteristics common to many of them. Of the kits included in the survey, the vast majority target online banks and auction Web sites such as PayPal, Bank of America, and eBay.

Marco and his colleagues also discovered that the kits themselves are frequently designed to defraud inexperienced phishers through vulnerabilities and back-door mechanisms in the kits. To prevent detection by suspicious phishers, the authors of such kits use various methods, from ones as simple as diverting a phisher's attention with misleading source-code comments to code hiding and obfuscation techniques.

One audience member asked whether the discovery of back-door mechanisms in the phishing kits was a result of the survey or served as its inspiration. Marco explained that such functionality was not expected initially and was discovered early on while analyzing one kit in particular. This led the researchers to develop the infrastructure used for their survey to identify similar functionality in other kits. Another audience member inquired about those phishing kits described in the survey without back-door mechanisms and asked what motivation people have to freely distribute such kits. Marco reasoned that in some cases the back-door functionality may have been removed by discerning phishers before installation.

- ***Towards Systematic Evaluation of the Evadability of Bot/Botnet Detection Methods***
  Elizabeth Stinson and John C. Mitchell, Stanford University

Elizabeth Stinson began by posing a question: "Is there a way to systematically evaluate the evadability of a [botnet]

detection method?" She went on to explain that as researchers continue to develop various botnet-detection methods, concerns about the evadability of each method (i.e., a botnet's ability to evade detection) invariably arise. The purpose of Stinson's work is to analyze how to accurately and objectively evaluate these concerns. Elizabeth was quick to acknowledge that evadability is not the only significant factor to consider when evaluating a detection method; however, it serves as a consistent litmus test to asses both utility and practicality across various detection techniques.

After reviewing some basic background information on bots, Stinson presented a framework developed by herself and co-author John Mitchell for measuring the evadability of botnet detection methods. Central to this metric are two costs: implementation complexity and effect on botnet utility. Implementation complexity is a qualitative measure of the effort to which an attacker must go to alter its bots to evade detection; the latter cost embodies an attacker's net reduction in botnet utility as a consequence of successful evasion of particular detection techniques. Stinson next discussed leading botnet detection methods, current evasion tactics, and, using their evaluation framework, the related costs. This evaluation led to a number of suggestions for improving existing and future detection methods.

One audience member wondered whether botnet detection is difficult in practice. Stinson explained that difficulty in detecting botnet activity is contingent upon a number of conditions including, among other circumstances, the perspective of the observer and the design of a botnet's command and control structure. Another audience member added that in enterprise environments in particular, detection and removal of even a handful of bots is often given high priority because of concerns of data theft and that, in general, current automated detection methods are imperfect.

- ***Reverse Engineering Python Applications***
  Aaron Portnoy and Ali-Rizvi Santiago, TippingPoint DVLabs

Portnoy and Santiago discussed the exposure of program structure inherent to programs written in dynamically typed program languages (e.g., Python and Ruby) and spoke about their experience leveraging such exposure to reverse-engineer Python binary applications. Owing to late-binding, applications written using dynamically typed programming languages often contain object metadata that is not typically present in statically typed programs.

In their presentation, Portnoy and Santiago demonstrated how this information can aid disassembly, decompilation, code object modification, and arbitrary instrumentation of Python applications. Using such techniques, they developed an application called AntiFreeze that is capable of visualizing and modifying Python binaries. Portnoy and Santiago demonstrated the value of AntiFreeze by presenting a case study reverse engineering a commercial Python application: Disney's Pirates of the Caribbean Online. Using AntiFreeze, the presenters were able to quickly and meaningfully modify the application, an MMORPG, granting their avatar otherwise unobtainable capabilities.

As the adoption of Python and other similar languages grows, Portnoy suggested that developers of dynamically typed applications should be wary of making assumptions about the privacy of their application logic, given the ease with which such programs can be reverse-engineered. Santiago suggested a number of possible approaches to mitigate exposure—for instance, modifying the Python interpreter. The presenters have publicly released AntiFreeze as a Google Code project. The URL is http://code.google.com/p/antifreeze/.

- ***Exploitable Redirects on the Web: Identification, Prevalence, and Defense***
  Craig A. Shue, Andrew J. Kalafut, and Minaxi Gupta, Indiana University

Internet users may notice that when contacting a Web site, their browsers are, on occasion, automatically redirected to addresses other than those explicitly provided. This is frequently done to seamlessly track user behavior, display moved content, and correct common typing mistakes in domain names. Under some circumstances (e.g., phishing attacks), attackers are able to exploit this redirection behavior to direct users to untrusted and malicious Web sites via links that appear superficially benign. These open redirects and their exploitation were the subject of Craig Shue's presentation.

One particular goal of Shue's research is to develop heuristics that automatically identify open redirects on the Web. Doing so allows Shue and his colleagues to measure the prevalence of such phenomena and provides an opportunity to mitigate their abuse by attackers. To evaluate their technique, they evaluated a large number of links for potential redirects. The links themselves came from three distinct data sets, each representing a different perspective on typical Internet usage. Of the three data sets, one consisted of links from the most popular Web sites (overall and by category) according to the Alexa Web Information Service. The other two data sets were composed of links from sites visited by members of Shue's Computer Science Department as recorded by their DNS queries and from the DMOZ open directory project. An evaluation of the researchers' techniques using these data sets yields positive identification of redirects in more than 58% of all tests.

Next, Shue proposed a number of approaches to reduce the opportunities for exploitation of open redirects. For each approach Shue detailed both positive and negative aspects, highlighting the challenges to protecting users from exploitable redirects. Some audience members inquired about the seriousness of this threat in light of other common system and network attacks. Shue and other members of the audience expressed the opinion that although such comparisons can be made, they are generally less beneficial than finding ways to mitigate or eliminate these threats.

PAPERS

Summarized by Joshua Mason (josh@jhu.edu)

- ***Modeling the Trust Boundaries Created by Securable Objects***
  Matt Miller, Leviathan Security Group

Matt Miller presented his work on automatically discovering data flows between trust boundaries in the Microsoft Windows operating systems. Trust boundaries are divisions between privilege levels on a system (e.g., different user accounts or user versus administrator privileges). Discovering paths of data flow between privilege levels allows software auditors to audit only those sections of code where vulnerabilities might actually lead to privilege escalation attacks. Using Miller's method, the auditor can quickly and automatically discern the relevant attack surface.

Miller's technique employs Microsoft's concept of a securable object to find the relevant data flow paths. A securable object is merely an abstraction for various system resources, including processes, files, registry keys, and so on. Each of these securable objects has a security descriptor that defines a series of access control lists. Monitoring these objects both dynamically and statically allows an auditor to discover those objects that allow complementary operations between access levels on the same object. For example, if a file can be written to by a given user and read from by the administrator, the file acts as a communication channel between the user and the administrator.

In addition to granting the ability to identify these privileged communication paths, the implementation of dynamic securable object monitoring allows an auditor to collect data on running systems that will allow auditors to identify paths of communication that actually occur. Merely discerning a user's ability to write to a given executable and an administrator's ability to execute it does not give any evidence that this actually occurs in practice. So, by letting a real machine run and collecting data over time, Miller is also able to discern data flow paths that are likely to occur.