Bug Hunting: The Seven Ways of the Security Samurai

Iván Arce, Core Security Technologies

The burgeoning bug population has enhanced public awareness about security. Here, the author outlines common bug hunting methods and techniques for actually finding bugs.

In 2001, the CERT Coordination Center (www.cert.org) received reports of 2,437 software security flaws in widely used software. This marked a significant increase over previous years and mirrored the findings of several other security-tracking agencies (see, for example, reports at http://www.nipc.gov/cybernotes/2001/cyberissue2001-26.pdf and www.securityfocus.com).

The effect of this burgeoning bug-finding fever has permeated the world in very interesting ways, ranging from the development of software programs that exploit vulnerabilities to increased mainstream press coverage to heated debates in the information security community over how to disclose findings. Of course, the increase in bugs also gave the technology industry itself a stream of bad publicity—and fodder for aggressive marketing campaigns.

Despite all this, little has been said about the actual bug finding process itself. As the "Myth vs. Reality" sidebar describes, the practice is shrouded in misinformation. Although the general public is well acquainted with terms like "hacker," "bug," and "virus," neither they nor many information security professionals themselves know how bug hunters find vulnerabilities or what systematic techniques they use. Here, I'll offer an overview of that process.

Bug hunting basics

You might think I'm letting the cat out of the bag by disclosing the obscure and seemingly magical set of bug-finding tricks used daily by a small, elite cadre within the larger information security community. Sadly, there is no such bag of magical spells to disclose. Although bug finding requires technical skills, experience, and an investigative (and often paranoid) mindset, it is far from being an obscure art approachable only by enlightened individuals.

To systematically find bugs, individuals do need

- common sense (to know what to look for),
- dedication (to spend endless hours poking through software code), and
- a bit of luck (to find meaningful results).

Also helpful are a touch of arrogance, a handful of tricks and tools, and considerable social skills for effective teamwork. In fact, the required qualities don't differ much from those a typical human being needs to live well in modern society.

For our purposes here, I define bug hunting as a systematic process in which one or more individuals try to find security flaws in a predetermined set of "technologies," including software products, hardware devices, algorithms, formal protocols, and real-world networks and systems. Constraints on the practice might include time, resource availability, technical expertise, money, work experience, and so on.

The goals of bug hunting also vary and are sometimes contradictory. A person might hunt bugs, for example, to

- perform technical due diligence on a given technology;
- assess the security of an organization's information technology infrastructure;
- incrementally increase a given component's security;
- identify and fix potential holes that might endanger organizations, communities, or countries; or
- find new ways to break into systems and networks for various legitimate or illegitimate reasons.

The best bug hunting approach includes a methodology or guideline that minimizes effort and optimizes results. Nonetheless, the literature offers little guidance and few standardization or best-practices efforts are under way (one of the few is OWASP's Web Application Security Testing Framework at www.owasp.org/testing/).

Bug hunting: The seven ways

Despite the lack of formal documentation, common techniques and methodologies exist for hunting bugs. Here, I outline seven that we regularly use at Core Security Technologies.

Source code audit

As the building block for a software program or device firmware, source code is highly valuable to bug hunters, as is

Myth vs. Reality

The absence of available information about how real people find bugs in real-world systems has led many industry people to create myths convenient for their own purposes, ranging from marketing strategies to technological elitism.

MYTH: Only very bright, knowledgeable people can find security bugs. Bug finders are the elite celebrities of the information security world.
REALITY: You don't have to be a genius to find a security bug. The most important bug hunting factors are the ability to put past experiences into practice and a willingness to labor for long hours, often with no results. That said, to find bugs systematically and quickly, it helps to have considerable security knowledge, a streamlined mental process, and an efficient team to get directly to the root of problems.

MYTH: There is always a shady (if not illegal) motivation behind the search for security bugs. Bug hunters are black-hatted crackers trying to find ways into our networks and personal information. Or, conversely, bug hunters always mean well and are working for a better, more secure world.
REALITY: Security bugs are often accidental, and the people who hunt bugs have varied motivations. For example, looking for bugs in a software product is a good way for consumers to assess the product's security before buying it, or for testers to identify weaknesses overlooked in the development process. Finding security bugs in a network architecture is required in certain security consulting offerings; it is part of day-to-day security maintenance work and mandatory for improving system security.

On the other hand, bug hunters from company Y might be tasked with finding vulnerabilities in company X's software and disclosing the information so that their own competing product looks better on the market. There are also individual bug hunters who seek fame or a good job, and of course, there are also those who seek to commit crimes of property or malice.

MYTH: The person that finds a security bug knows everything there is to know about it—including how to exploit it.
REALITY: More often that not, security bugs are found by people who lack a clear understanding of the bug. Understanding why a bug exists, how and when it manifests itself, and how to fix it properly requires much additional effort that does not necessarily relate to the actual bug-finding process. Also, more often than not, the bug finder knows neither exactly how to exploit the bug nor the exploitation's potential scope or side effects. Most bug finders are unaware of the full, global implications of their discoveries.

frequent interaction with the code's development and maintenance team.

To do an audit, bug hunting teams carefully read the complete source code and identify and document each bug they find. The first and most effective approach is to find and analyze known bad programming practices within the code. (For an overview of good and bad programming practices, see the Secure Programming Standards Methodology Manual at http://uk.osstmm.org/spsmm.htm.)

In the next (and often overlooked) stage, bug hunters reread the source code to obtain an in-depth understanding of its design and purpose, which helps them find security bugs that are intrinsic to design and implementation.

The source code audit method, also known as RTFS ("read the fine source"), is well suited for experienced bug hunting teams, but does not require extensive knowledge of all of the underlying technologies.

As the amount of code or the software complexity increases, source code auditing can become painfully time-consuming and resource intensive. A viable shortcut is to select and audit a representative set of code, such as the source code of the software's basic components or critical modules. Also, having the development team provide support and answer design and implementation questions can accelerate the process and almost always guarantees the best results.

Reverse engineering: Debug & disassembly

When the source code is not available, bug hunters must take a different approach. If the software is relatively small or simple, reverse engineering might work.

To find bugs using the reverse engineering approach, two things are crucial: software debugging and disassembly. *Debugging* entails actively monitoring the software's execution to understand its functions and how it carries them out. Software developers regularly use this technique to find and fix software bugs (security-related or not) when a program misbehaves or when a bug manifests itself during program execution, but they've detected no problems with the source code.

Bug hunters apply the same idea. They select program inputs and follow their execution path, thereby gaining an understanding of the input's progression and identifying security bugs. The team can change inputs to force the program into other execution paths that might yield new findings. However, it's impossible to debug a program and follow every possible execution path. (I leave proof of this as an exercise to the reader.)

Using this debugging technique, teams commonly find security bugs arising from invalid input processing or mishandling exception conditions (errors) caused by invalid inputs.

Disassembly entails obtaining and then analyzing source code from an executable image of the software product. How is this different from a source code audit? Disassembly yields source code in assembler, the lowest-level programming language for the software's microprocessor architecture. Furthermore, the code can be very complex and generally differs from the high-level programming language source code that produced the executable image (though it performs the same functions). Also, development team members' input is less important, because they developed the software in high-level language and thus have less understanding of how the software's features, functions, and components relate to disassembled code.

To disassemble an executable image into assembler source code, the bug hunter uses a disassembler tool that interprets the executable image format and translates it into assembly code for the architecture it runs on. The tool must therefore "know" about the underlying microprocessor programming features. Some disassemblers offer additional features, such as recognizing disassembled code constructs that correspond to commonly used constructs in high-level languages or the underlying microprocessor architecture.

Armed with feature-rich debuggers and disassemblers, an auditor can engage the bug hunting process in a promising way. Compared with source code auditing, reverse engineering is more resource-intensive and requires the most technically skilled bug hunters to achieve good results. Success also depends on the target software's complexity and size; disassembling a large application or major chunks of an OS might be too resource intensive. Finally, reverse engineering is illegal in the US under certain conditions as a result of the Digital Millennium Copyright Act, which was born of at best innocuous and at worst flawed attempts to prevent the understanding and disclosure of trade secrets or intellectual property. (The DMCA is available at www.loc.gov/copyright/legislation/dmca.pdf; for details of its aftermath, see http://sci.newsfactor.com/perl/story/13187.html.)

Reverse engineering: Network traffic

When source code is unavailable, or when the target technology is large or complex (such as a proprietary operating system) or interacts with other network components, bug hunters must understand the technology's global interactions and identify problems therein. To do so, they can use a network sniffer—a software program that captures all the packets traversing the network off the wire—to spot possible flaws in the communication mechanism.

Bug hunters need not always fully understand the communication protocols involved. Modifying and replaying the captured traffic or generating spurious traffic can reveal security bugs. Searching for bugs based on component communication analysis typically requires an expert bug hunter who understands networking protocols and their use in real-world scenarios. It requires less low-level expertise at the platform or OS level.

Black-box security testing

In some cases, bug hunters have no information about how the target technology works and cannot access it for reverse engineering. This might be the case with firmware stored in tamper-resistant devices, network appliances, or even software products. Here, bug hunters can treat the target technology as a black box—a device that accepts certain inputs and produces a set of outputs. Basically, they know the device's purpose, but not how it works.

But not even a black box operates in the vacuum: it has a defined environment such as an OS, computer network, or physical location. By modifying different inputs or environmental conditions, bug hunters can review the outputs and determine if the black box is operating securely—that it's doing what it is intended to do and nothing else.

For example, say our bug hunters have a Web server as a black box. They'd first send it certain inputs (URL requests) and obtain certain outputs (Web pages). Next, they'd send it malformed requests or alter the server program's network or OS. The Web server's outputs should be the same as expected under normal operating conditions; if not, the server likely has a bug. The bug hunters would then determine whether the bug was a security bug.

The key is to alter the environment and inputs to reveal security bugs, which typically result from bad design and implementation practices. Here, bug hunters' expertise and experience come into play: They look for known security bugs, such as buffer overflows, format string bugs, race conditions, cross-site scripting bugs, authorization and authentication flaws, and denial of service bugs. (For a more detailed classification, see http://online.securityfocus.com/cgi-bin/vulns-item.pl?section=help&id=4146.)

Black box analysis typically requires considerable experience and time from the bug hunting team, though it does not usually require deep technical skills. An exception to this is when the target technology is a very specialized device that performs an obscure task.

Brute force

Automated tools offer a variation on the black-box testing approach. The idea here is to automatically try every possible bug-exposing input to the target program. This requires less expertise and relies on good testing tools. However, it might miss fundamental design flaws or semantic attacks (bugs related to the inputs' meaning, rather than the inputs themselves). Also, bug hunters might end up with innocuous bugs—security flaws that are present in the technology, but unimportant in standard use.

Top-down analysis

Top-down analysis starts from a high-level overview of the target technology and drills down to details once the team suspects specific weaknesses. With this approach, bug hunters first gather and analyze documentation and all available design materials for the target technology and any related technologies. This might include design specifications, protocol specifications, white papers, formal descriptions of underlying cryptosystems, or Request for Comments issued by the Internet Engineering Task Force.

From such documentation, bug hunters identify both theoretical problems in the target technology's design and known problems in its related technologies. They then look at the technology in more detail to confirm or deny the existence of the theoretical bugs.

This approach requires experience in the information security field, a high-level understanding of many technologies, and a more academic approach to their problems. Its advantage is that it yields results quickly; however, it can lead to many dead ends and identify theoretical flaws that don't exist in the real world.

Information gathering

Finally, the quick and lazy—but often highly rewarding—approach is to search for bug reports or symptoms on all possible sources, including Internet search engines, user mailing lists, newsgroups, product documentation and release notes, and product-support knowledge bases. Doing so gives bug hunters hints and sometimes specific information about pos-

Real-world bug hunts: A sampling

As the following examples show, bug hunters have applied the methods and techniques described here in the real world with great success.

The OpenBSD security audit

In the summer of 1996, the OpenBSD project began a systematic source-code audit looking for bugs, using several of the techniques described in this article. In less than a year, the teams found and fixed thousands of security bugs, and discovered and formalized whole new security bug classes. This systematic bug search, along with the incredibly quick response time on bug fixes, earned OpenBSD a reputation as the most secure general-purpose OS available. A detailed account of their bug hunting process is at www.openbsd.org/security.html#process.

"Netscape engineers are weenies"

In April, 2000, independent security expert Rain Forest Puppy published a report detailing how Alf Serer found the string, "Netscape engineers are weenies!" within a Microsoft Windows NT 4.0 OS component. The string was part of the authorization mechanism to access server-side components of Windows-based Web sites, and posed a security risk to many Web servers. The finding triggered a storm of e-mail exchanges in the security community debating whether or not it was definite proof of the flaw's existence. Given the nature of both the bug and the string, the finding generated significant press coverage. Claims that the finding did not expose a security bug prompted Gerardo Richarte and Alberto Soliño to do a quick audit of the component in question. This soon revealed the existence of yet another, more serious security bug. Throughout the process, several individuals around the world applied reverse engineering techniques and procedures, unveiling the problem within hours. A full account of the incident is at http://archives.neohapsis.com/archives/ntbugtraq/2000-q2/0035.html.

MySQL authentication bug

In October 2000, Core Security Technologies (CST) published a report detailing a security vulnerability in the widely used MySQL database engine. Using a time-constrained source code audit of the database engine and assembly-line teamwork, we produced extensive details of the security problem and how attackers might exploit it. The complete report is available at www.corest.com/common/showdoc.php?idx=125&idxseccion=10.

SSH CRC-32 insertion attack

In June 1998, CST's Ariel Futoransky discovered a basic design flaw in the communication protocol for the Secure Shell (SSH) software package while reading the product's design and protocol specifications. SSH provides strong authentication and secure communications over insecure networks and is widely used to achieve secure log-in to remote computers over a network. A few days later, Futoransky and CST's Emiliano Kargieman audited the SSH package's source code and confirmed the bug's existence. They worked on a fix and sent it to SSH developers, who incorporated it into virtually all software packages that implemented the SSH version 1 protocol. This is a good example of top-down analysis, coupled with peer auditing of source code. The bug report is available at www.corest.com/common/showdoc.php?idx=131&idxseccion=10.

SSH "deattack overflow" bug

In February 2001, information security firm Bindview published a report detailing Michal Zalewski's findings describing an obscure bug in CST's fix to the SSH insertion attack discovered in 1998. The bug existed in all SSH software packages that incorporated the 1998 fix. This is a good example of how two rotating teams can discover bugs overlooked or introduced by each other. Bindview's problem report is available at http://razor.bindview.com/publish/advisories/adv_ssh1crc.html.

sible security bugs; they can then attempt to verify their existence in the target technology.

For example, a Web search for "product X" and, say, "SIGSEGV"—which indicates an abnormal program termination in a Unix OS and is symptomatic of certain security bugs, such as buffer overflows or formatting string bugs—will yield hundreds of results. The bug hunter can then refine the search for specific security-related terms. Next, the hunter might actually reproduce the bug or research the reported problems to verify their validity.

This method provides quick results and requires no great expertise or experience in the security field. However, the approach is not a good long-term strategy: it is unsystematic and can often lead the bug hunter on blind searches for security bugs without any guarantee of timely success.

Bug hunting: Methodology

The techniques above deal with the technical details of bug hunting. However, good methodology is also needed if bug hunters are to find bugs quickly and with minimum effort, and to do so consistently over time. To outline such a methodology, we must consider the constraints on the bug hunting process, including time limitations, available security expertise, technology resources, tool availability, and legal implications.

In addition, there are several criteria for any systematic a bug hunting effort.

- *A clear definition of the target technology*. Example definitions include: a default installation of Microsoft Windows 2000 Advanced Server OS for the Intel platform, running on specific hardware; the Apache Web server version 1.3.22 standard distribution for Linux, including all the default modules; a proprietary application in its production environment, including its current configuration and all the related software and hardware.
- *Process documentation*. Everyone involved in the process must understand what is being done, what is known about the target technology, and what directions they might follow during the process to more easily find security bugs. Documentation is always crucial, but is particularly so if the project involves many individuals with different expertise and experience, or people working in physically separate locations.
- *Results documentation*. Documenting the results and clearly stating what was left out of the process provides useful information for future work.
- *Diversification*. Relying on a single technique or person to hunt bugs is a bad idea and will produce limited results. Always use a mixture of techniques and a team of two or more people.

There are several options for organizing a systematic hunt, depending on time and resources. Following are five of the most common approaches.

Lone ranger mode (one person or more)

The most basic approach for an exhaustive security bug search is to dedicate an individual or team to the process on an ongoing basis. In this scenario, each person's sole responsibility is to find security bugs in all target technologies. With a team, each member covers all technologies independently, and exchanges his or her findings with the other members. This process is obviously best suited for long-term efforts; it does not take into consideration strict time constrains or expectations for short-term results.

Time-constrained peer audit (two to three people)

In this approach, two or three people audit the same technology together, sharing resources and tools. They might sit together at a computer, with one auditing while the others watch and provide advice and fresh ideas. This methodology predates similar Extreme Programming techniques (www.extremeprogramming.org) and ensures that the target technology is examined from different viewpoints and thus that fewer bugs will be overlooked. This method is particularly useful when time is tight and team members have similar skills and experience.

Assembly-line teamwork (two or more people)

When you have people with varying degrees of security expertise or many technologies that must be audited quickly, it's a good idea to form a bug hunting team of people with specific skills. Such a team might include

- a seasoned security auditor with general knowledge about various technologies;
- a highly skilled person with in-depth, low-level knowledge of specific technologies, such as assembler for x86 or reverse engineering;
- an academic with theoretical cryptography knowledge; and
- a QA analyst or software developer.

Given this team, the security auditor might do a cursory technology audit to spot possible weaknesses, then pass the technology to the person most suitable for dealing with that weakness. That person would further expand on the initial audit, identify bugs, and pass it on again. The process ends when a security bug is positively identified and the team has proof of its existence. Meanwhile, the high-level security auditor has already moved on to the next technology and put another assembly line in motion.

Although this method might miss several bugs in a specific technology, overall, it results in the discovery of many security bugs.

Tournament (teamwork)

For a more comprehensive effort, you can form several bug hunting groups (organized as in previous examples), select a set of target technologies, and assign a subset of them to each group. You can then implement a tournament-like process, encouraging groups to find as many bugs as possible in a given time period. The group that finds the most, wins the tournament (and maybe a prize).

Although this is a highly efficient way to find numerous security bugs in a short time period, achieving well-documented and researched results requires the commitment of many people, considerable logistical effort, and strong project management during the tournament. You should also establish rules and procedures to score a "bug finding" in advance and ensure they are followed throughout the process.

Rotating teams

In this approach, several teams look at the same software in rotation. The first team does an audit, reports bugs, and gets them fixed, then the second (and possibly a third) team does the same. Thereafter, the first team starts over again and the cycle is repeated until an established time limit is reached. Or, if it is an ongoing part of the software development process, the cycle is repeated indefinitely. This process guarantees that almost all bugs will be found and fixed; what one team misses, another will catch. Another advantage is that teams can audit and (if necessary) amend the previous team's bug fixes.

Using a systematic approach, you can find many security bugs in a short time. Doing so provides tangible benefits to system security and is a necessary step in any short-term attempt to close security gaps.

Some people argue that bug hunting would be unnecessary if the technology itself was better built, and that that's where more serious effort is needed. However, past history and experience have shown that security bugs will always exist, no matter how good the security QA program. Given this, we must integrate both strict security QA programs and systematic bug finding processes as we develop and maintain the technologies that will drive our world into the future.

Acknowledgments

I thank all the people who contributed with thoughts and ideas for this article, all those who encouraged me to write it, and my colleagues at Core Security Technologies who, during the past five years, provided me the raw material for this article, and also showed almost unbearably great technical knowledge, security expertise, and passionate dedication to their work.

Iván Arce is CTO and cofounder of Core Security Technologies, an information security company based in New York. Since starting the company in 1996, he has managed bug hunting teams at CST and been involved in the discovery of hundreds of security bugs. Contact him at ivan.arce@corest.com.