Software as an Exploitable Source of Intelligence

Dr. David A. Umphress
College of Aerospace Doctrine, Research, and Education (CADRE)

Security goes beyond the traditional notion of hacking into a piece of software. Software, even without being installed or used, can reveal compromising information, thus providing a security risk. This article outlines four software exploitation categories that should be considered before a software product is released.

Our security practices tend to separate data from software. We think of data as *content*, meaning, something of operational value that can be exploited by an adversary. We tend to think of a computer program as something that performs tasks and manipulates data, not as something that has inherent informational value in itself. But we cannot escape the simple fact that software exists; it is a collection of computer instructions and supporting data. As such, it is a *thing*, something that has the potential to be broken into, taken apart, scrutinized, cannibalized for parts, or otherwise used for purposes not originally intended.

It is exactly this potential for useful information that makes software attractive for *software vulnerability attacks*. Such attacks start with software in the same form as it would be given to a legitimate user and then subjected to a number of static and dynamic tests in order to reveal compromising information. Software vulnerability attacks include the traditional notion of hacking, where a computer is broken into over a communication network, but also encompass a broad range of other tactics that view software as a source of intelligence data. Attacks need not necessarily be launched against software systems that are operational. They may be more subtle in that the attacker has physical possession of the software and is examining it for vulnerabilities under circumstances that are unobserved.

Forms of Exploitation

Software vulnerability attacks draw from the work of software security, reverse engineering, design reclamation, and software testing to answer the question, *what does the product reveal about itself?* Unless explicitly corrected otherwise during the development phase, a piece of software has the potential to be open to four general categories of exploitation: intrusion penetration, intellectual property penetration, component penetration, and context penetration. Offering a prescription for how to identify vulnerabilities in each of these four categories is difficult – and beyond the scope of this article. However, recognizing that software is open to exploitation beyond the traditional notion of hacking is a necessary first step toward developing software processes which address holistic security.

- **Intrusion penetration** is the act of gaining illicit use of software. Someone engaging in intrusion penetration would seek to discover whether the software limits user access to functions and, if so, how securely the software deals with determining authorization. Such a vulnerability attack would analyze the software for how it authenticates users, how – or if – it encrypts data, what software features it allows users to perform, and so forth. The ultimate goal of intrusion penetration is to masquerade as a legitimate user, thus gaining access to as much functionality and data as the software offers, including, if possible, access at the level of a *super user*.
- **Intellectual property penetration** is the discovery of business rules, classified information, and protected computations encoded in software. Consider, for example, a software system that processes telemetry data from an infrared sensor in order to detect the heat signature of a missile launch. The software may well be considered unclassified when stripped of data relating to the sensor and telemetry stream, but the computer instructions themselves reveal how the data is processed, and can thus inadvertently reveal information that could be exploited. Consider also a personnel database. The structure of the database alone, absent any data, could reveal insight into what information is maintained on personnel, organizational structure, maximum size of organizations, and so forth.
- **Component penetration** addresses how the software might be used outside the context for which it was written. This form of penetration begins by discovering the individual components in the software, much as an electronics engineer would identify distinguishable components on a circuit board. Software components could then be extracted and reused in other software applications. Alternatively, software components could be extracted and replaced with substitute components that have the same interfaces but provide different functionality.

  In the first case, an adversary could obtain the use of a critical software element – such as a decryption algorithm or communication module – without having to re-create it from scratch. In the second case, the replaced component could report on the inner workings of the software, thus giving an intruder a further toehold into how the software might be further exploited programmatically. Indeed, it is not infeasible that, under certain circumstances, an intruder could intercept software being transmitted over a network and replace components with ones having nefarious purposes.
- **Context penetration** extrapolates what is known about the software under scrutiny to larger systems it may be a part of. For example, the network communication traffic that a client receives, processes, and transmits can reveal the purpose and function of its corresponding server.

It should be noted that none of the exploitation categories noted above are trivial. Most must be carried out manually though laborious examination; however, minimal attacks can uncover surprisingly revealing information, as evidenced by the following:

- **Intrusion penetration.** A software component known as a *wedge* was inserted to intercept communication between two existing software components of a large military research package. The wedge did not interfere with the functioning of the software; however, it displayed the content of data being transferred from one component to the other as the software was being executed. Through trial and error, the wedge was successfully placed at a point that revealed the license key needed to decrypt user information.
- **Intellectual property penetration.** A cursory examination of software used to track finances of a large organization revealed the toll-free access phone number, login name, and password to the organization's communication hub. While this information was not sufficient to gain access to financial information, it provided a security hole through which

an adversary could masquerade as a business unit and submit bogus data on financial transactions.

- **Component penetration.** The automatic software update mechanism was extracted from a system used to manage membership information for a national non-profit organization and was transplanted to another piece of software. In the original system, the software referenced a local XML file containing a URL and the current version numbers of the individual software modules. Using the URL, it obtained from the Web an XML file that described the most recent version of each software module, compared it to the current version, downloaded updated modules, and installed them.
- **Context penetration.** The components and configuration files of the previous system were well-enough named that their purpose was self-evident, simplifying the task of isolating the automatic update modules and adapting the configuration files for a totally different application. Although the server end of the update mechanism for the newly adapted application had to be constructed from scratch, it could be modeled after the original's client and server information interchange.

Relevance to Today's Technology

Why are software vulnerability attacks relevant today? Harvesting information from a software artifact has heretofore been difficult and time consuming. Most products of the past have been delivered as a collection of binary executable machine instructions that mask the structure of the product and do not give easy insight into how the product might be exploited. Modern technologies that have come into heavy use in the past five years (specifically Java and .NET) have reshaped the product landscape by permitting software to be encoded using generic programming instructions. Instead of having the computer's hardware directly execute the instructions, a special program reads each hardware-generic instruction, verifies that it will not violate the computer's security policies, and carries it out as if it were part of the computer's instruction set. Since the encoded instructions are intended to be executed on any hardware platform, they must carry with them information on software module structures, data types, etc. This approach allows software to be written once and run on many different hardware platforms, thus providing on-demand delivery and installation of software to networked computers (often a necessary piece of electronic commerce and *enterprise* computing). The disadvantage is that it does so at the expense of making the software more open to analysis.

Conclusion

Understanding software is a source of intelligence is vital to anyone involved with developing or distributing software. Of the four exploitation categories, only one, intrusion penetration, is typically examined in any depth for a given software product. Paying attention to the other categories will become more important as time goes on due to the increasingly crucial role software plays in today's economy. It is in any organization's interest – both industry and military – to minimize the amount of information exposed by software independent of it being installed or executed.◆

About the Author

David A. Umphress, Ph.D., is an associate professor of computer science and software engineering at Auburn University. He has worked over the past 25 years in various software development capacities in both industry and academia. Umphress is also an Air Force reservist, currently serving as a CADRE researcher, Maxwell AFB, Alabama. Umphress is an Institute of Electrical and Electronic Engineers Certified Software Development Professional.

Department of Computer Science and Software Engineering
107 Dunstan Hall
Auburn University, AL 36849
Phone: (334) 844-6335
Fax: (334) 844-6329
E-mail: david.umphress @auburn.edu