Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
2004		**00-00-2004 to 00-00-2004**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
Software as an Exploitable Source of Intelligence	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **CADRE/AR,Director, Airpower Research Institute,401 Chennault Circle,Maxwell AFB,AL,36112-6428**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**	**Same as Report (SAR)**	**2**	

CADRE Quick-Look

Catalyst for Air & Space Power Research Dialogue

Software as an Exploitable Source of Intelligence

Lt Col David Umphress

Problem/Issue. Software, even without being installed or used, can reveal information that we do not wish to be disclosed.

Background. 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. But, we can't 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.

Software vulnerability attacks (SVA) start with software in the same form as it would be given to a legitimate user, then subject it to a number of static and dynamic tests with the intent to yield a description of how the software might be open to unintended use. SVAs are different from the traditional notion of network hacking, where a computer is being broken into over a communication network. SVAs are more subtle in that the attacker has physical possession of the software and is examining it for vulnerabilities in circumstances that are unobserved, and potentially unobservable.

SVAs 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?" In general, SVAs seek information leading to three forms of software exploitation: intrusion penetration, component penetration, and intellectual property penetration.

- *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, etc. 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."

- *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.

- *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

The College of Aerospace Doctrine, Research and Education (CADRE) *Quick-Looks* are written by military defense analysts assigned to Air University (AU) and beyond are available electronically at the AU Research Web Site (https://research.au.af.mil) or (http://research.airuniv.edu). Comments are encouraged. Send to: CADRE/AR, Director, Airpower Research Institute, 401 Chennault Circle, Maxwell AFB AL 36112-6428 or e-mail: cadre/arr@maxwell.af.mil.

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, etc.

- It should be noted that none of the exploitation categories noted above are trivial. Many penetration attacks may be computationally infeasible to carry out; however, minimal attacks can uncover surprisingly revealing information, as evidenced by the following:

- A 30-minute examination of software used to track finances of large organizations 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 by which an adversary could masquerade as a business unit and submit bogus data on financial transactions.

- A software "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 key needed to decrypt user information.

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 executable machine instructions that mask the structure of the product and don't 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 won't 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 structure, 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.

Possible Solutions:

- Software Threat Assessment Lab. Form a center for evaluating the vulnerability of software. Such a center would receive software in the same form as would be delivered to users. The lab would try to penetrate the software within a given timeframe, and report back to the originator any observed vulnerabilities.

- Software Offensive Operations Center. Form a center for collecting and penetrating software from adversary groups. The goal of the center would be to serve as a collection point for technical intelligence as well as for advising on how to exploit vulnerabilities.

- Increased awareness education. Catalog software vulnerabilities as they are found, together with potential defenses. Distill and publish common vulnerabilities/defenses as software development security "best practices."

- Digital software signature defense. Participate with industry in formulating standards for signing, authenticating, and verifying software.